UXPin

Responsive Web Design Best Practices

Advice, Tutorials, Case Studies

UXPin

Responsive Web Design Best Practices

Advice, Tutorials, Case Studies

Copyright © 2015 by UXPin Inc.

All rights reserved. No part of this publication text may be uploaded or posted online without the prior written permission of the publisher.

For permission requests, write to the publisher, addressed “Attention: Permissions Request,” to hello@uxpin.com.

Index

Understanding the Case for Responsive Web Design **7**

What is Responsive Web Design? 8

Why Responsive Web Design? 11

The old way is fading out 13

Case study: Plusnet 15

Takeaway 16

Mastering the Responsive Design Philosophy **18**

Understanding Responsive Design 19

Responsive Design Properties 20

Ask Yourself: What's In It For the User? 23

How do you decide what's essential? 26

Start Thinking Content-First 30

Personalization: Let the User Drive 31

Case Study: Virgin America 34

Conclusion 37

5 Useful Responsive Design Layouts **39**

Mostly fluid 39

Column drops 40

Layout shifter 41

Tiny tweaks 41
Off-canvas 42
Conclusion 43

Responsive Navigation 44
Touch Targets 45
Feedback 47
Consistency 48
Clarity 50
Useful Navigation Patterns 54
Additional Examples 62
Tutorial: Responsive Navigation Drawer 66
How to add “a come back” 72
How to show the navigation drawer on swipe to the left and hide it on swipe to the right 74
How to make “left-side” navigation drawer 75

Responsive Typography 78
Reading Distance 79
Establishing Responsive Text Size 81
Line Height 84
Line Length (Measure) 86
Conclusion 87

Responsive Images **88**
Choosing the Right Format for the Job 89
Squeezing Every Byte 93
Compression Services 93
Code Considerations 99
Implications for UI Design 101

The Technical Side of Responsive Design **103**
Media Queries Are the Technical Heart of Responsive Web Design 104
Compressing Resources for Faster Load Times 109
HTTP requests 110
Takeaway 111

Putting It All Together: The Mobile-First Workflow **113**
Question Your Assumptions 113
Collaborate From the Beginning 114
Embrace Mobile-First Design 115
Mobile-First = Content-First 117
The Mobile-First Process 118
Going forward 127

Ben Gremillion is a Content Designer at UXPin specializing in responsive design. Previously, he was a Design Writer at ZURB. He has earned an Adobe Certification and knows CSS, HTML, regex, PHP, MySQL, and other impressive-sounding acronyms. He also builds and maintains a CMS for webcomic artists, and participates in bi-annual NaNoWriMo challenges.

Jerry Cao is a content strategist at UXPin where he gets to put his overly active imagination to paper every day. In a past life, he developed content strategies for clients at Brafton and worked in traditional advertising at DDB San Francisco. In his spare time he enjoys playing electric guitar, watching foreign horror films, and expanding his knowledge of random facts.
Follow me on Twitter

Zack Rutherford is a UX Design Writer at UXPin. His work has been published in UX Mag, awwwards, The Next Web, and Speckyboy.

Understanding the Case for Responsive Web Design

Responsive Web Design, or RWD for short, is an oft talked about and frankly invaluable discipline in the context of today's design industry. Moreover, it's a term that grows more robust in meaning as different ideas and aspects of device usage are introduced to the public. A few short years ago, it was enough for a design to render attractively in four or five different preset viewports.

This is no longer the case today.

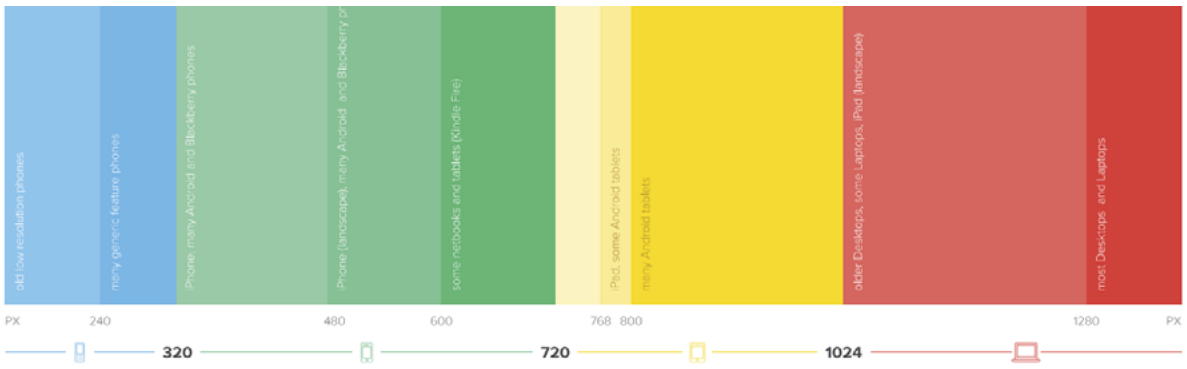


Photo credit: Responsive Design Cheatsheet

As mobile browsing outpaced desktop usage in 2014 with statistics predicting a continuing increase, a website must have a mobile strategy in place in order to succeed. Moreover, the increasing variety

of devices and screen sizes require designs to be more flexible than ever before.

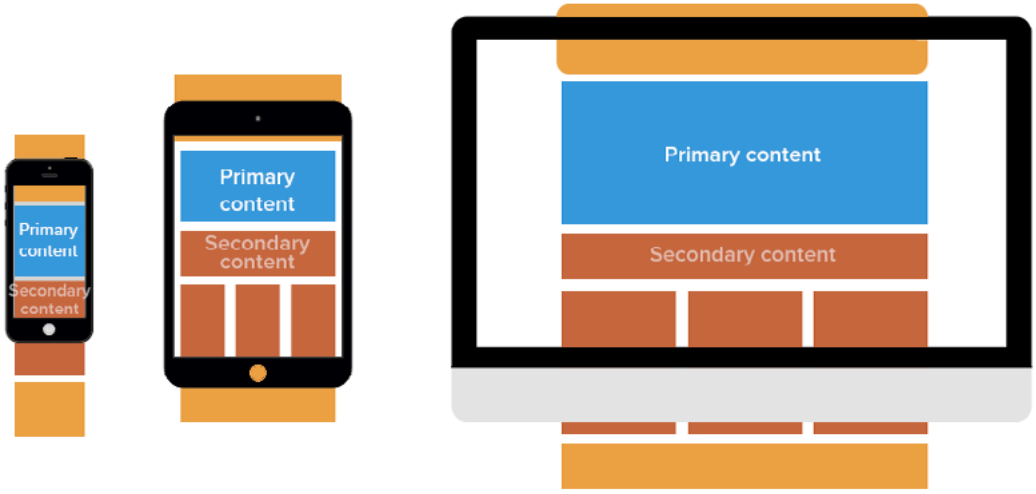
And while there were once multiple options for serving mobile users, such as m-dot sites or heavy reliance on mobile applications, the most relevant research to date shows that these solutions just aren't as consistent or as cost effective as a responsive design.

In this book, we'll guide you through the ins and outs of making your work not just respond well to different screens, but a delight to use across a range of browsing experiences. Beginning broadly with RWD as a concept, we'll work our way through to specific examples of how and why to use certain methods to achieve the most attractive and consistent design regardless of the user's device.

By the time you've finished reading, you should have a solid understanding of what RWD is, when, where, why, and how you should apply specific techniques according to the needs of a certain situation. No matter what your level of familiarity with RWD right now, you will be able to glean additional insights by delving deeper into the topics and techniques described within this book.

What is Responsive Web Design?

As the name suggests, Responsive Web Design responds to the user's viewport, device, or platform. Essentially, it's a design that can attractively acclimate to any screen resolution.



Responsive web design is beginning to go beyond this definition, but it is still traditionally recognized by three primary concepts:

1. Fluid grids – An arrangement or organization of content lacking a fixed width, which is composed of intersecting vertical and horizontal lines. This grid is used to structure content in a predictable and consistent manner by adjusting the size and positioning of the elements contained within.

2. Flexible images – Any visual content or media that adjusts to fit a user's screen size. Usually these are images placed within a flexible grid that has the max-width CSS rule applied at 100%. In layman's terms, this prevents images from being larger than the grids in which they're placed while also allowing images to resize without sacrificing any of their aspect ratio. To avoid slow loading times, designers can compress the images' resolutions when displayed on smaller devices. Another method is to set the width as a percentage of the width of the page as a whole. This assures a consistent differential between the size of the image and the page size, meaning that no matter how the page width

is manipulated, the image will always appear sized as a percentage of that width.

3. Media queries – A CSS3 module that controls how the styles within the media attribute are applied. Different styles are applied according to a number of device specific qualities such as screen width, height, orientation, etc.

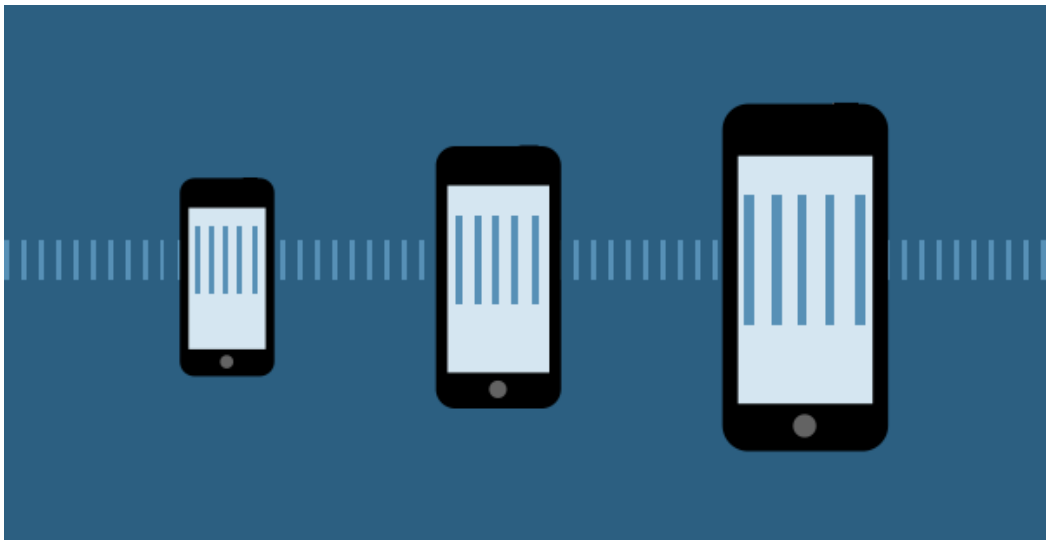
These cornerstones form the foundation of what we understand as Responsive Web Design. There are, however, other associated ideas which have become more or less synonymous with RWD. Mobile-first approach, progressive enhancement, and unobtrusive JavaScript are all closely entwined with the principles of responsive design.

It will be interesting to see how much the term comes to symbolize in the future. Right now, people are already confusing the term with the more breakpoint centered, Adaptive Design. Recently, there has even been talk of expanding the definition to include taking advantage of unique device capabilities (such as specific commands on a touch screen).

The definition, perhaps not ironically, is quickly becoming as fluid as the concepts it describes.

Why Responsive Web Design?

The diversity in devices used to browse websites is only growing larger, and it has become an expectation that users take for granted when exploring their preferred destinations on the web. Your design must respond to the user's viewport if you want to deliver even a baseline acceptable experience.



More to the point, RWD is the preferred method of making a website more accessible by virtually everyone:

- Users don't like dealing with a redirect, and downloading a dedicated app is a hard sell when they're dealing with limited mobile disk space.
- Multiple websites for a single domain become costly to maintain, and difficult to update as well. When you add an extra codebase, you also add more maintenance cost in the long-run. You'll either need to deal with twice the work or use a server-side solution, both of which are more expensive than a responsive or adaptive site.

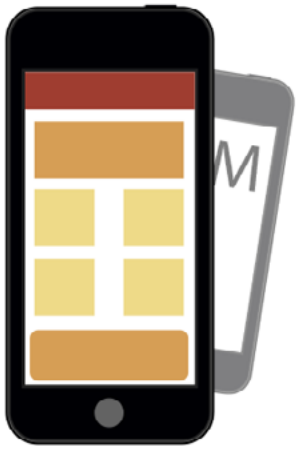
A well implemented responsive design, on the other hand, addresses all of those concerns. If a single set of images, grids, and the like are automatically resized to fit the user's viewport, then the UX is by definition more consistent than switching between two separate sites. And if you only have a single codebase to maintain and update, it's not going to cost as much money nor take as much time.

The advantages of RWD are immediate, and well documented:

- Increases your audience, sales and conversion rates. According to a study by the Aberdeen Group, RWD sites achieve 11% more conversions than non-responsive sites on average.
- Forces you to prioritize content for each viewport (especially smaller screens), which inevitably makes for a stronger site.
- Truly future-proof since you don't need to obsess over every new device screen size
- Improves your SEO: Google officially recommends responsive sites.
- Gives users uniform quality – no one wants to be a second-class citizen.
- Enables you to be detail oriented, which is great because (if you're a good designer) you care about details.

The old way is fading out

M-dots, or separate versions of a site for mobile users, are a dying practice.



M-dot sites like “m.samplesite.com” vs. “www.samplesite.com” provide a different browsing experience for users on different devices. When the goal is consistency, it becomes easy to see why this isn’t the ideal solution. Still nothing is quite as clear as statistical data, and there’s plenty of evidence for the decline of M-dots.

- **Pure Oxygen Labs reports that last year M-dot sites fell 20%**, from 79% in 2013 to 59% in 2014, while responsive and adaptive (dynamic serving) sites rose 37% collectively.
- **Users visit the full site anyway** – Web Performance Today’s research showed that about a third (35%) of users choose to go to the full site if given the option.
- **Users spend more time on the full site** – The same research states 5.5 times longer. They also calculated that 79% of revenue from mobile sales came from users on the full site.

- **SEO/Google trouble** – According to Google's own guidelines, responsive and adaptive sites will likely rank better. Not using an M-dot is an automatic boost in SEO.
- **Redirect time** – While M-dot sites load faster in theory, the extra time of redirecting from your full site to the M-dot (unless the user types the M-dot's URL) is unnecessary. Alongside the other drawbacks, is it worth it?
- **Mobile devices aren't a single screen size** – It's ironic that what was once the greatest strength of M-dot sites is now its greatest weakness. M-dot sites are designed for a specific screen size, but mobile devices range from 320×240 for some smartphones up to 768×1024 (and beyond) for tablets. It just doesn't make sense to serve the same layout to all those screens.

Bottom line: M-dot sites are a bad idea because they cost more and create inconsistent experiences.

There are numerous examples of sites who make the switch to RWD being exponentially more successful after a relaunch. Let's take a look at an example from a case study performed by Google on the British Internet Service Provider, Plusnet.

Case study: Plusnet

When Plusnet's analytics began showing a significant increase in mobile traffic to their website, they began considering a major change in their web offering.

As Google's case study shows, the numbers were impressive and they started growing quickly. More precisely, the amount of mobile traffic was rising a percentage point (in relation to total traffic) every month. Once it became obvious that the trend would continue for the foreseeable future, the decision-maker's at Plusnet concluded that they had to invest more in a mobile strategy.

They opted for a responsive redesign because they, "...felt responsive web design was the best solution for developing content that will work across all devices on one platform rather than on multiple sites, making it as future-proof as possible given today's technology."

Photo credit: Think With Google

Along with the implementation of RWD, Plusnet decided to dedicate more of its budget to mobile marketing. Even though their site is far from being a pinnacle of visual design, the single act of switching to responsive was powerful enough to yield impressive business results. Time to conversion was slashed nearly in half (40%), and sales via smartphones and tablets increased ten times over in consecutive years. Now, the new responsive site's mobile conversion rate is higher than the old site's conversion rate was for all devices combined.

These results aren't atypical when switching from having no mobile strategy to implementing RWD. When Time Magazine relaunched their responsive site, the number of unique visits to their homepage increased by 15%, with time spent going up by 7.5% and the mobile bounce rate decreasing by 26%.

Good responsive design is good business. And if you want to be a good designer, you must always design for the right business reasons.

Takeaway

The conclusions are easy to draw.

More people are using mobile now than ever before, and the number of mobile users is continuing to grow. More than half of the population has a mobile internet service subscription, and by the end of 2020, that number is projected to grow by another billion people.

Without an effective mobile strategy in place, site owners are missing out on a swarm of exposure. And if businesses need responsive design, then designers better master the responsive skillset.

But, as we've discussed so far, simple media queries, fluid grids, and flexible images aren't enough to provide the best possible UX. What's required of designers now and in the future will be an open mind and a flexible attitude to match that of the evolving web.

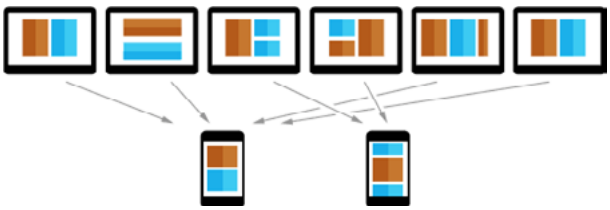
Responsive design is more than a game of technical adjustments to a layout. It is a complete reimagining of the way in which content is delivered. The much more difficult challenge is strategizing how to deliver the right content to the right user on the right device at exactly the right time.

In the next chapter, we'll discuss the philosophical implications of what it means to make designs truly responsive.

Mastering the Responsive Design Philosophy

Philosophically speaking, responsive design only begins with fluid grids, flexible imagery, and media queries.

What makes a design truly effective across mediums is the implementations *after* the foundational layer is laid. Once you've developed the scaffolding for your design, it's time to build a structure around it that can react to new technologies, different viewports, fresh types of content, and the changing needs of the user.



That's what your philosophical aim is, and so that's what you have to bring to your practical applications. An idea of fluidity and an attitude well-suited to the embrace of inevitable change.

We'll explore how to understand responsive design and its properties, then dissect some live examples to see the practical application.

Understanding Responsive Design

To embrace responsive web design is to acknowledge that the web is a fluid medium.

Content changes over time, whether it's frequent updates within a design framework (e.g. a template) or occasional redesigns (changes to the brand/updates to technology). Either way, you'll have to accept the fact that your work is going to be adjusted.

This presents an interesting dilemma. How exactly are you supposed to future-proof your work?

You can't anticipate the conditions in which users view your work. It seems like the entire design industry reinvents itself every 5 years or so already.

Consider:

- Browsers change, new devices emerge.
- Viewport sizes and shapes change with new product releases.
- And let's not get started on automatic updates... they usually don't even work that well.

A truly responsive design is one that truly takes all of this into account, and builds allowances for future developments into the infrastructure. This can get confusing in a hurry.

How are you supposed to make allowances for something that you can't predict? The secret is to be open-ended in your expectations and to find creative workarounds when confronted with constraints you'd normally think of as immutable.

To elaborate, that means you must endow your designs with fluidity, inside and out. Which includes all of the following:

- Sensible layouts that scale logically to fit the design's content.
- Imagery that doesn't degrade.
- Keeping user needs at the forefront of your decision-making.
- Prioritizing content by designing around it.
- Providing freedom of movement and greater control, regardless of device.

Quite the list. Let's take a look at how it's accomplished.

Responsive Design Properties

People's sense of proportion remains constant. When looking at a screen (and for the foreseeable future we can assume that these will come in rectangular shapes) a user is always going to expect a height which rationally corresponds to the width.

Therefore, your screen height must be proportional to the screen width.

For example, given the same amount of content, as a viewport becomes narrower it will also require more vertical space to fit the desired content. Unless you're expecting words to gradually disappear within a sentence as the screen shrinks, it's a good idea for the cell of your grid to get taller as the screen gets skinnier.

One thing that will help you accomplish this in reference to typography is the use of **ems or rems**.

- **Ems** are the relative units of measure which can be used to change the size of typography in relation to the element in which it's contained.
- **Rems** on the other hand are similar, but they conform to the size of the Root element, as in the top level of HTML, instead of the element containing the typeface.

When coding in fixed pixel sizes, you'll find your layouts will be restrictive and they won't scale very well across a wide range of platforms. But implementing ems or rems means you'll always have a text size that renders at a constant proportion to screen size, or more specifically, the size of the block of content in which it's contained.

You need more than just your typeface to scale up or down attractively. That's where **SVGs (or Scalable Vector Graphics)** come in. These are images that aren't composed of pixels, but rather made of paths.

A "path" is basically a set of encoded instructions for the renderer, that determines whether the path is a line, or a curve, etc. This means

that as screen resolution increases, the image doesn't lose any quality because it's still being rendered along the same paths. Zoom in or out on an SVG as much as you like and you still get the same image without all the blocky pixelated artifacts you'd get from a raster image.

Finally, it's important to remember that blocks of content help layouts adapt to different screens. Responsive designs that set content elements into blocks to be rearranged and vertically stacked are a great solution to a viewport that will inevitably be adjusted.

To recap, common RWD elements which will enable you to provide a future-proof design should include all of the following:

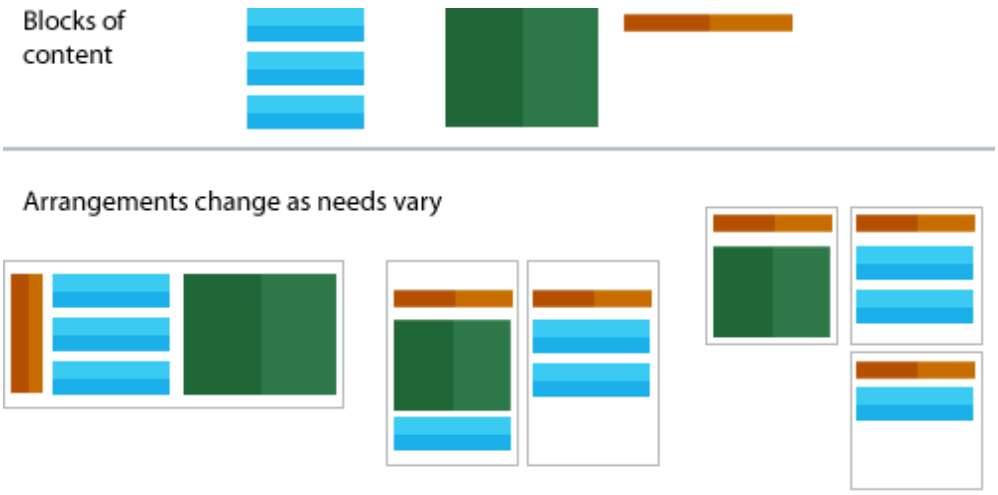
- Screen height is proportional to screen width. Given the same amount of content, the narrower a viewport is, the more vertical space it will require.
- Layouts benefit from relative units. Fixed pixel sizes restrict layouts to certain devices.
- Vector-based art scales are preferable to raster or pixel based. In the latter cases, art will lose quality if required to expand. In short, it's easier to make images smaller than larger.
- Blocks of content help layouts adapt to different screen sizes.

Now that we've covered some characteristics that match the responsive philosophical bent, it's time to move on to the real star of your designs: the user.

Ask Yourself: What's In It For the User?

It's called responsive design because it must respond to the user's circumstances. The devices and browsers they use, the content and utility they're looking for–every design decision is a bespoke element tailored to the user's needs.

When prototyping your designs, remember that content is the reason they're visiting the site. Knowing this is good practice for any site, but it's especially helpful when you're deciding what to show on a tiny screen.



Of course, responsive design is about much more than just reacting to shrinking screens.

You've heard that content is king and you must design for mobile devices first. Both the mobile-first and content-first approaches are fine. They're helpful ways to frame your thinking. It's always easier to scale up than it is to scale down. However, neither are very useful without **a purpose-first approach**.

Without a clear purpose in mind, mobile and content are, well, meaningless.

What gives design its purpose? Certainly there are business objectives that must be achieved, but sales are only possible when a business' goals are aligned with those of the user.

In general, users want to accomplish a goal or gather information. They might want to better manage their bank accounts or a get the latest news. Businesses, on the other hand, generally want users to act in a certain way:

- Buy a product.
- Sign up for a newsletter.
- Donate to a cause.
- Agree to or accept a certain point of view (as is the case with political campaigns, for example).

As a responsive designer, it's your job to prioritize where user and business goals overlap. What serves both audiences? When starting a new project always ask yourself:

- What benefit(s) does a product give users?
- What will the business do with collected emails, and what useful information will the end user get in return?

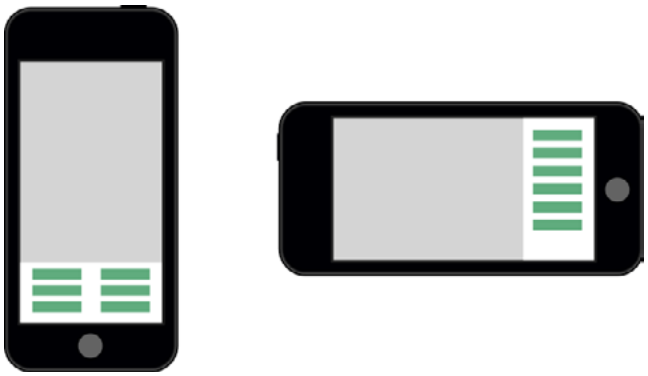
Answering questions like these will help prioritize what the design conveys at any given screen size.

For every page you put together, think about each element and whether it would matter to the user if it disappeared. If you conclude that users won't miss an element, then you're better off discarding it. The less elements on the screen, the better (and faster) the layout adapts to devices.

And while you're discarding and disregarding, don't forget about context. Device and browser capabilities like location awareness, lighting conditions, possible gestures and screen orientation should all be in the back of your mind while working.

Here are a few examples of functions that require you to think beyond screen:

- If geolocation is enabled in a user's device, you can use it to serve up different localized versions of content, as is often the case with retailer websites. But you'll also need to create a layout that doesn't fall apart with different versions. For example, set up a back-end condition that shrinks the typography from 16 to 14 points if a certain number of characters are exceeded (to accommodate all the content). You'll also need to test content extremes (e.g. little content vs. long content) to see how the interface holds up.
- Screen orientation requires different navigation treatments due to changes in thumb placement. For example, in portrait mode for a mobile viewport, you may fix the navigation at the bottom. But when you switch to landscape mode, you'll need to revise the navigation to a sliding navigation menu (or fixed horizontal sidebar).



- Mobile devices often include gesture controls such as pinching to zoom. You'll need to set the appropriate maximum zoom. We'd recommend no more than 200% zoom (from a technical standpoint, that means setting the HTML to "maximum-scale=2.0").

How do you decide what's essential?

Even though it's nice to add elements that provide value, you still only want the minimum amount of elements per page.. This can be tricky. You need a system to decide whether or not a design needs one element or another.

To accomplish this, follow these steps:

1. Invent a tagline and consider the design implications'
2. Write down a list of what makes each page/view unique. Put these features front and center and discard any elements repeated elsewhere.
3. Start by laying out the content for the smallest viewport, then expand your layout for larger views. Create a minimum of 3

viewports (smartphone/tablet/desktop) and no more than 5 (for the sake of maintenance).

4. As you prototype, try the site/app yourself, then add elements that help you get around, while subtracting those you find superfluous.

5. Integrate branding into the content: colors, typography, and icons. Make sure to add enough that your designs are visually consistent but not overloaded.

To clarify on that first item, taglines are extremely helpful when determining the importance of on-page elements, features or functions.

For example:

- **"The service for foodies on the go."**– In this case, geolocation is essential so users can find all the right locations. Remember the localized content considerations we discussed previously.

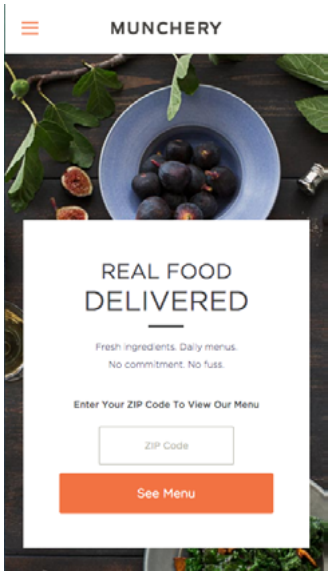


Photo credit: Munchery

- **"We make sharing photos easy."**– Consider a card-based layout with an infinite scroll since you'll need to organize large streams of visual content. Card-based layouts also work well for responsive design since they are naturally rectangular.

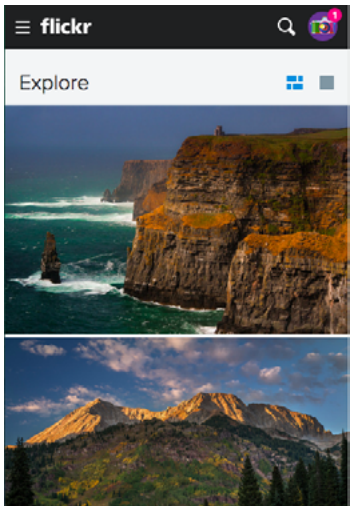


Photo credit: Flickr

- **"Financial advice that takes the pain out of paying bills."**– Minimalism is a must here. People want to find the info they want as quickly as possible and get out of the site. For the smallest viewport, present only a single clear log-in option above the scroll.

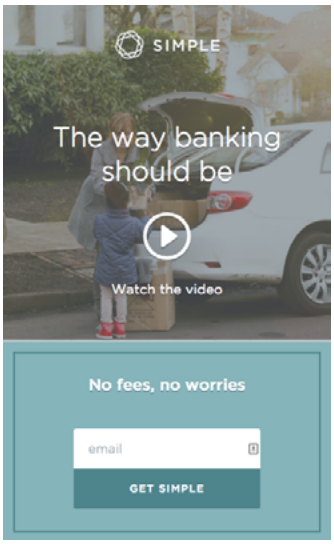


Photo credit: Simple

- **"The go-to source for all things digital design..."**– This is clearly a text heavy service. You'll need to lay out blocks of text in a flowing, easy-to-scroll manner. In this case, consider the mostly fluid layout to fulfill your purpose.

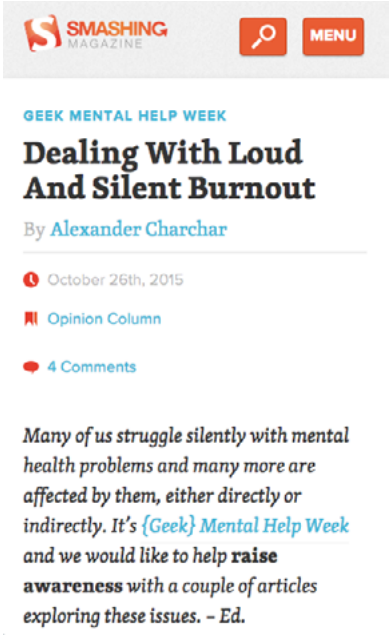


Photo credit: Smashing Magazine

One more thing to remember while prioritizing your content: if you need to add a lot of elements on a page, don't feel pressured to fit every feature at the top. You prioritize your content for a reason. As long as there's a clear hierarchy, you have plenty of room. Don't obsess over the fold. People will scroll.

Remember also that all your content can be used as branding. In other words, treat the information on your page as if it were a design element. Here are some guidelines:

- Avoid stock art unless it tells a story. Lean towards art that only works with the material at hand. If it could work with anything

else on the site – say, any of your blog posts – then it's too generic to be meaningful.

- Typography in content sets the tone. See also Typography.net.
- Don't design containers for content. While pre-made templates are usually visually striking, the danger is that they force you to squeeze content into premade areas. To stay focused on user goals, let the content dictate the layout.

Start Thinking Content-First

Get it in your head early: without content, design is bound to be flawed. Content is the foundation of your design choices. It's the information your users want, and the way it's displayed is essentially the bells and whistles. Your navigation, functionality, art, imagery, typography, and layout should all be constructed with the message of your content at the forefront of your thinking.

Photo credit: Stephanie Walter. Creative Commons.

- Take inventory of what content needs to be displayed, categorize it, and mandate a proper place for it within your website.
- Separate pages for each category of content. Similar content types can be contained on the same page as appropriate, but the main thing is to let the tone, theme, or topic of each piece of content determine the design of the page.
- Make a bullet point list of your content. Categorize and structure it in order to scaffold parent and child pages (starting with the smallest viewport). This exercise will help create your site's architecture, and inform your implementation.

Personalization: Let the User Drive

Once more (and this can't be stressed enough), the philosophy of responsive design is about far more than simple screen size. You're trying to build personalized experiences for your users based on the immediate context.

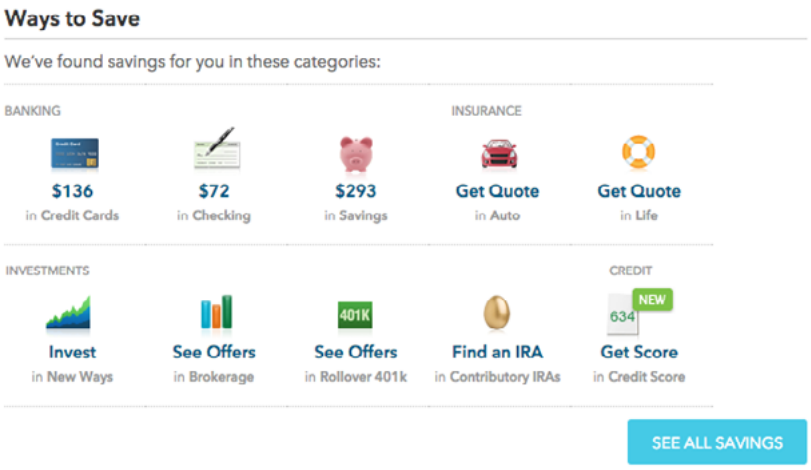


Photo credit: Mint

Part of this is leveraging what data you can gather, which these days is quite a lot. Be sure that your interfaces take into account the any previously logged data, such as:

- Search settings.
- Search history.
- Preferred location.
- Previous purchases.

Remember too though, as many marketers will tell you, it's easy to get bogged down with the details when dealing with large datasets. The more important consideration you can make is to let users control their browsing experience. They know better than you what's going on in their individual case. So make sure they can:

- Pinch and zoom – don't set "user-scalable=no" in the HTML
- Show/hide interface widgets as needed, perhaps make them disappear as they scroll.
- Tap to close widgets they don't need (such as secondary navigation).
- Adjust the brightness of the UI for well-lit or dimly-lit conditions. Will your color palette still be visually appealing at both extremes of screen brightness? Test your colors with the Contrast Analyzer and Luminosity Analyzer tools.

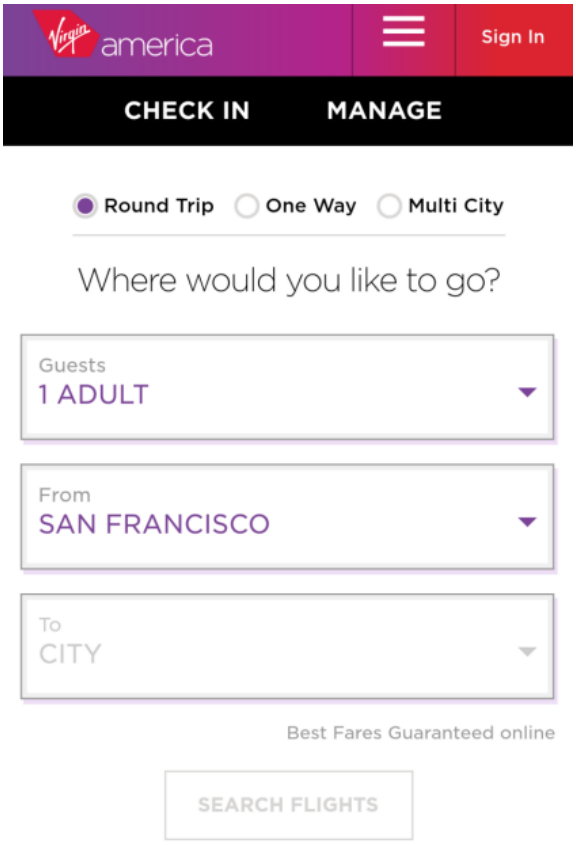
While current technology might limit the extent of personalization, you should certainly adopt the mindset as soon as possible. Remember

that responsive design is really just another way to describe content personalization.

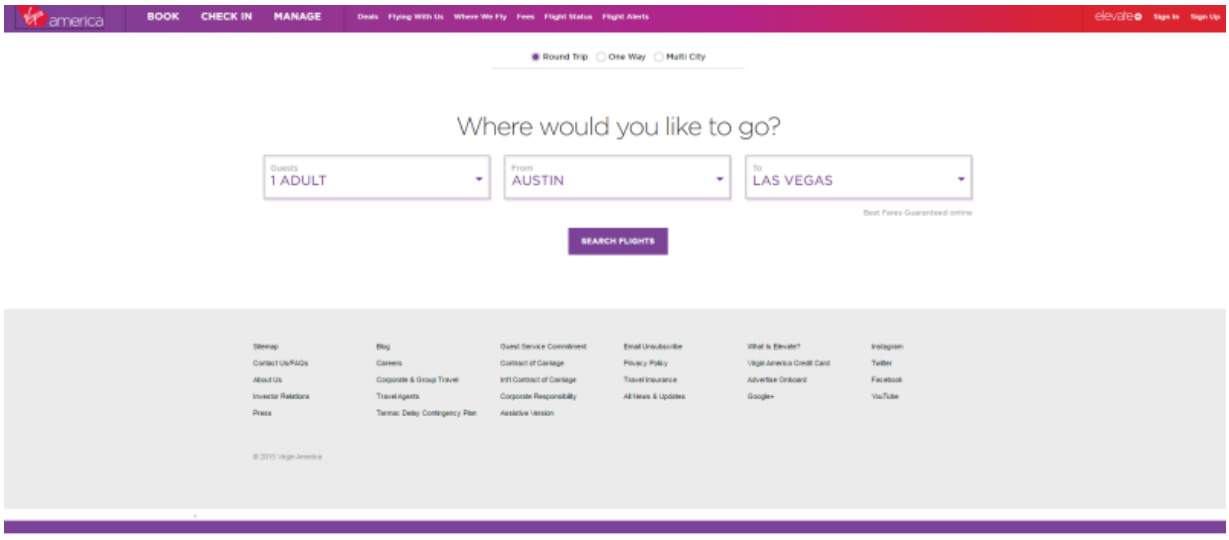
Now that you've got a good idea of the specific attributes of a successful responsive design, let's examine a real world application.

Case Study: Virgin America

Smallest Viewport:



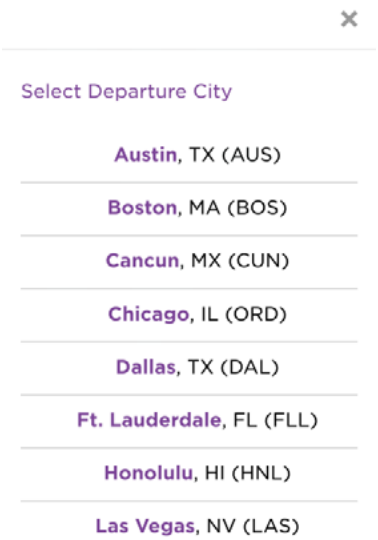
Largest Viewport:



Virgin America has a trendy reputation already, and this case study by Work & Co represents yet another step in the direction of responsive design savviness. Note how the home page puts the user's content needs front and center across multiple viewports.

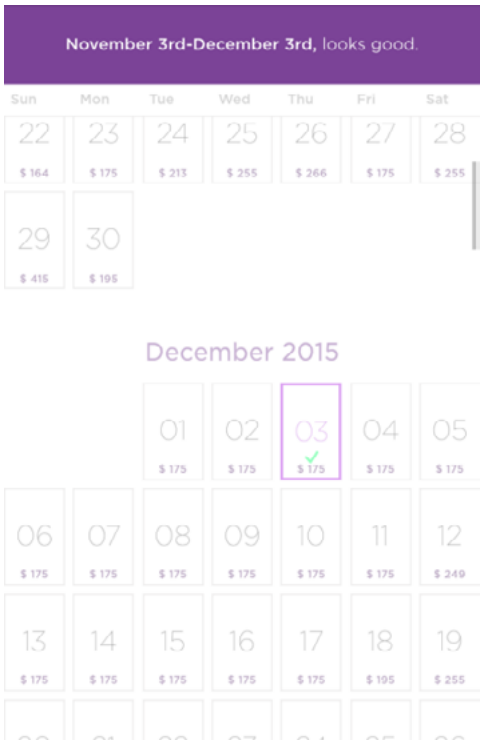
- The Call to Action is immediately presented, requesting a user's travel information
- Drop down menus accommodate the user's needs, allowing them to quickly enter in the number of seats they need as well as their destination.
- On a smaller screen, navigation options are sequestered to a full screen menu available at the click of the hamburger icon.
- The drop down items (Guests, From, To) aren't only available on the largest viewport (desktop). In the smallest viewport (smart-phone), the items are instead presented in a new window. The core task is unaffected by device.

Smallest Viewport:



Once you decide upon your departure and arrival dates, the interface even provides a fantastic bit of conversational feedback “[Date], looks good.” at the top of the page:

Smallest Viewport:



The user is in the driver’s seat from the first step of the booking process all the way through. The navigation is simple, straightforward, intuitive, and familiar across devices. What’s more, when in the smartphone view, the primary content immediately fill their viewports with a single touch.

Not only that, but by implementing this content service through a series of full screen menus, the responsive site doesn’t have to load any new pages. This saves time while simultaneously achieving the goals of the design. Namely, to increase online bookings.

Beyond personalized content service, the responsive site has a very measured and elegant approach to the design details:

- **Soft, subtle animations** – Communicates a sense of progress without overwhelming the senses.
- **Cool subdued color schemes** – Puts the user at ease while working through the often stressful process of planning for travel.
- **An auto scroll function which fully displays the next section of the page** – Clearly establishes hierarchy and makes for a consistent UX while scrolling.
- **Truly responsive flight booking** – Most airlines stick solely with apps, but Virgin recognized that they'd lose out on mobile bookings if they relied only on in-app purchases. An app plus RWD strategy allowed them to diversify their revenue stream and provide an amazing browsing experience to non-desktop users.

Philosophically speaking, this case study hits on many of the major points that add up to outstanding responsive design.

Conclusion

In conclusion, the philosophy behind responsive necessitates that designers not only have purposeful, scalable, and attractive content at every possible screen siz,e but also that they consider the context of the users.

This means making allowances and changes for the ways that certain users in specified locations will be likely to interact through geolocation, remembering their preferences, examining their search histories and other data related matters.

But more than that, it means allowing the user to take control of their interactions. Let them determine their direction and engagement. Your design should simply be there to guide them toward their own goals.

5 Useful Responsive Design Layouts

Google Product Director Luke Wroblewski recently described five broad categories of responsive layouts. Reiterated in a Google Developers post, these patterns give designers a head start when deciding how and why to set elements on a page.

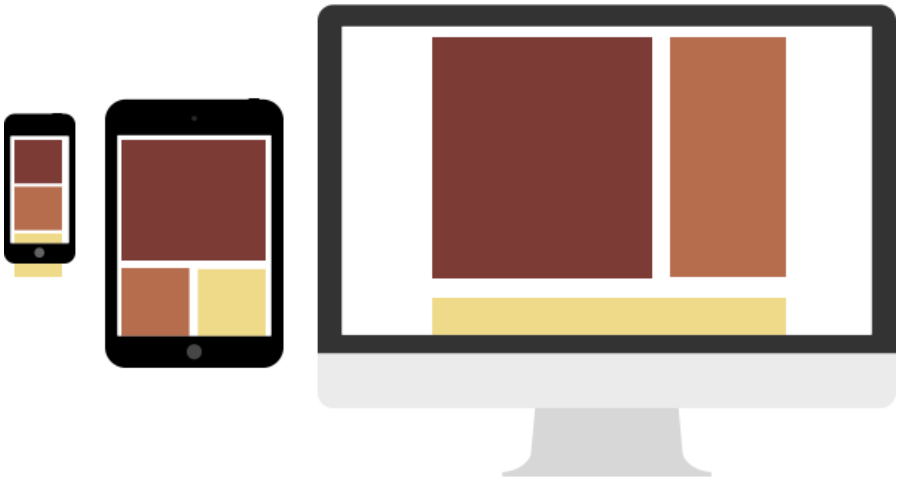
Remember that an "ultimate responsive layout" doesn't exist. Each of these options comes with its own merits, and the best choice all depends on the type of site you're designing.

1. Mostly fluid

   Luke's "mostly fluid" pattern reflows content dynamically within a fixed parent container instead of the browser's viewport. Parent containers – those that hold other elements – frequently use the CSS max-width property to keep from expanding beyond a certain width while fitting within narrow viewports.

   According to Google, this pattern only requires one breakpoint, assigned to the parent element, while the elements inside them reflow automatically as the parent resizes. This pattern works

well on sites with a diverse set of layouts among its many pages like marketing sites, ecommerce sites, news sites – anything with more than one type of content.



2. Column drops

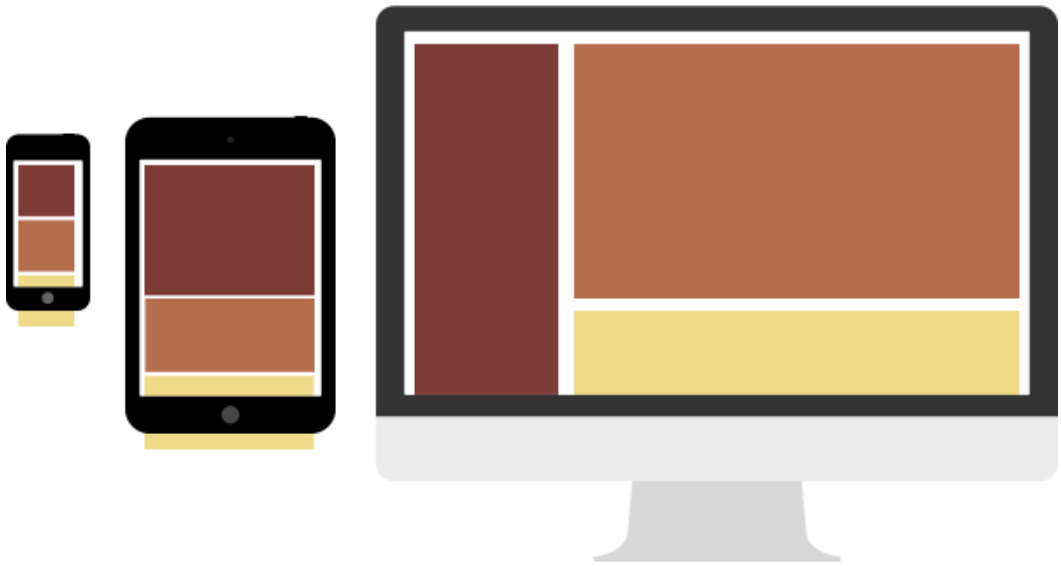
   Thinking mobile-first, Luke's "column drop" pattern turns a stack of columns on its side to take advantage of additional horizontal space in tablet and desktop-sized viewports.

   Unlike other patterns, this one tends to fill the available viewport regardless of size. This is ideal for responsively designed sites that users will view on a very wide range of displays.

3. Layout shifter

   Unlike the column drop pattern, Luke's "layout shifter" uses many breakpoints.

   Smartphone and tablet-sized viewports see a typical stacked layout, and the design grows more complex with wider browsers by rearranging content with a specific plan. This requires designers to consider each breakpoint range as if it were an independent design, but works well for visual-heavy sites like portfolios, agency sites, and photo galleries.



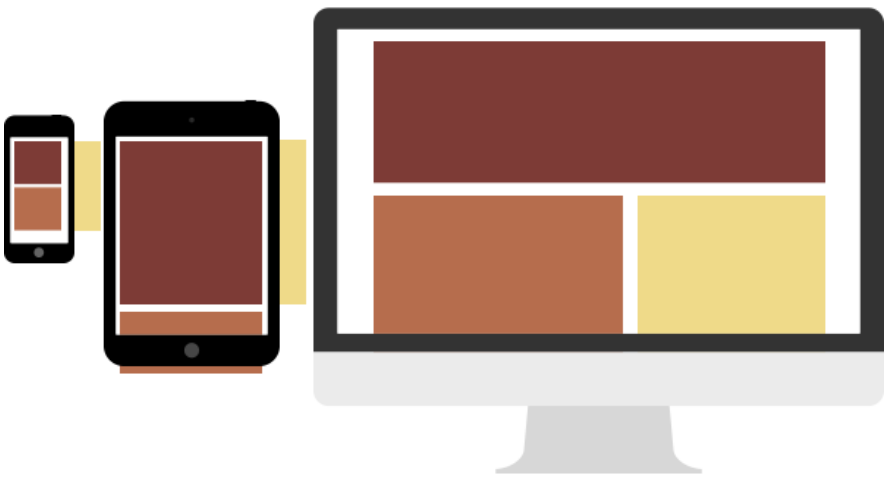
4. Tiny tweaks

   Luke's "tiny tweaks" pattern is the most straightforward strategy: content, margins and padding all subtly increase with viewport size while using the same layout. This low-maintenance approach best serves simple, single-column designs – or designers on tight deadlines with little budget for later adjustments.

5. Off-canvas

   Luke's "off-canvas" pattern pushes secondary content out of sight, literally outside of the viewport until called upon.

   Rather than stacking content vertically on small viewports, this pattern places them side by side with a trigger – typically JavaScript – that "shifts" adjacent panels into and out of the viewport. This is a real boon not only to navigation, but also to blogs that need to focus on large amounts of text and keep asides literally to the side.



Conclusion

When Luke Wroblewski researched these at the Media Queries website, he looked for broad, repeating trends among many different sites. He found that there is no one quick design solution that covers every situation.

And while exceptions to the above patterns may exist, these cover most situations that occur across the web. Designers can choose from these five patterns to give their layouts direction before they design – a real head start in their work.

Responsive Navigation

Every site needs links to their top-level sections or pages. Traditionally, designers accomplished this with increasingly complex navigation bars and drop-down menus that let users navigate an entire sitemap of material.

With space at a premium, users on mobile devices need new ways to get around a site. Dropdowns require a precise hand on a touch screen. As anyone with fat fingers knows, that is certainly a challenge for certain users.

When it comes to navigation, you must be extremely careful in your design choices. Navigation options have to be immediately available, unobtrusive, easily usable, and completely intuitive. You can't afford to miss any of these characteristics.

What other navigation patterns prove successful for responsive design? And what are some of the steps we should take to ensure our users have a positive experience when navigating on smaller screens?

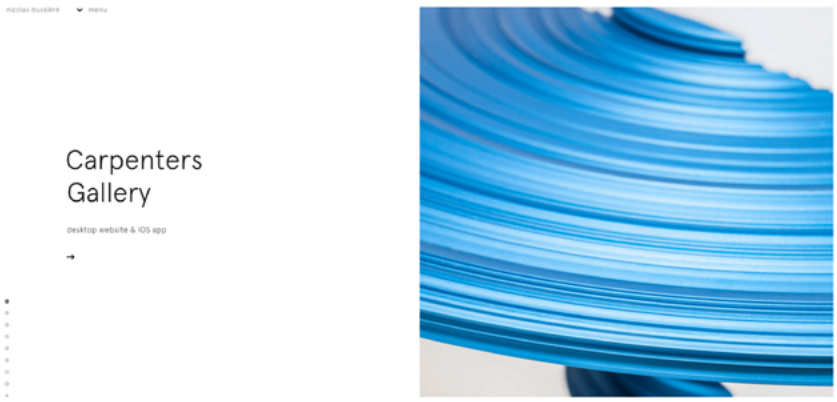


Photo credit: Nicolas Bussiere

In this piece, we'll explore web design best practices through a responsive lens, then explore these tactics in action with useful responsive navigation patterns.

Touch Targets

First and foremost: remember to make touch targets (buttons, calls to action, etc.) easy to hit. Know the importance of fat finger design.

1. Using the corners: On very small screens, the lower left corner is the easiest place for users to access, most often with their thumb. On tablets, which are held differently than phones, the top corners are prime touch real estate. When designing your small screen layout, place CTA buttons or other important links in that lower left corner to ensure ease of access. For mid-range screens, which is where most tablets will fall, place these important buttons and links in those upper corners.

2. Tap targets (like CTA buttons) must be adequately sized. A minimum of 44 points is usually sufficient.

3. Do not place items too closely together to avoid someone accessing the wrong item by mistake. Suggested spacing to avoid interface errors is a minimum of 23 points.

4. For smaller viewports, forget about hover states. On touch screens, there is no “hover”, so ensure that any information revealed this way for non-touch screens is still accessible. You can have a tap open up the menu that would have been revealed on hover. You could also design your interface so those sub-menus are available from the start, with no tap or hover needed.

5. Use natural interactions and create navigation that works well with common hand gestures, like swiping.

The only real way to define how effectively your buttons are sized is to test them exhaustively. Use different sizes and spaces throughout a prototype and start clicking things. See what’s easy and what’s irritating. Take a lot of notes and you’ll have a good idea of what works and what doesn’t.

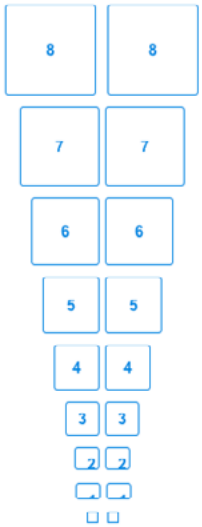


Photo credit: UXPin

Once you've iterated after testing the touch targets on yourself, test with at least 5 users and then iterate again if needed.

Feedback

Users also need feedback quickly. This goes double on a touch screen. Website visitors are notoriously impatient, mobile users even more so. If they don't think their actions are being processed instantly, they're more likely to navigate off your site. Don't make your users wait and guess.

Options for providing feedback when users are navigating include:

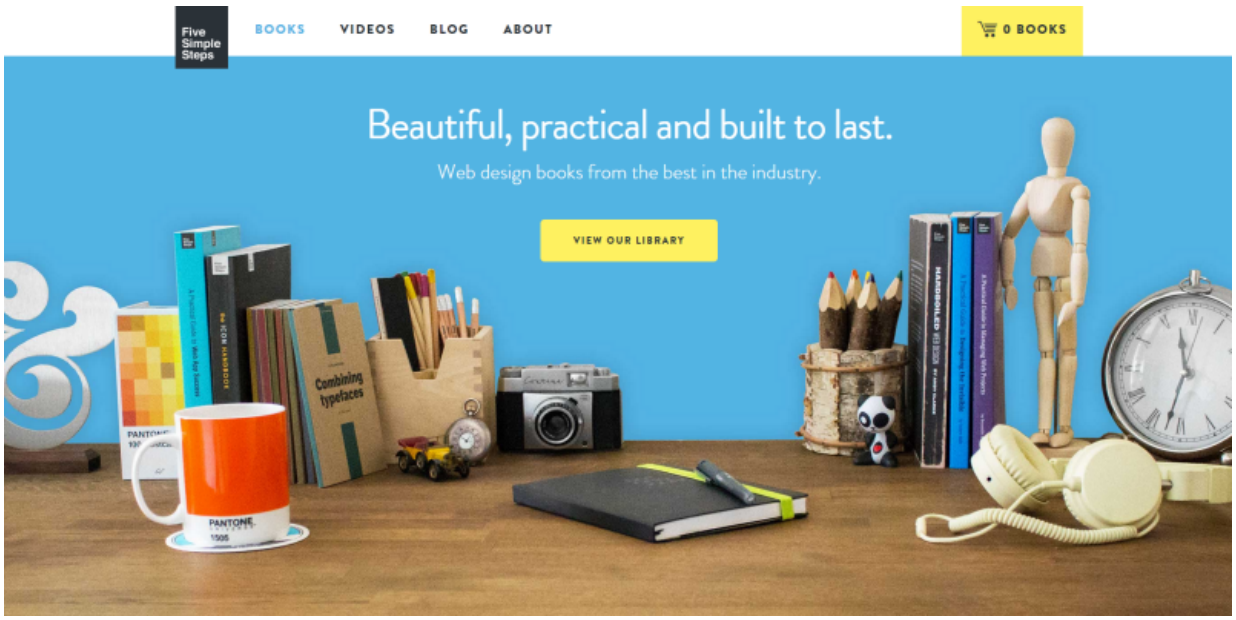
- **Animation** – A button changing color or shape is a great indicator of interaction. Remember to take the mobile-first approach: first sketch out each frame of the animation in the smallest viewport, then scale each frame up to larger viewports. Web designer Joni Trythall provides an excellent responsive animation walkthrough in her article for Designmodo.
- **Loading screens/graphics** – If you're serving an image-heavy site to your users, it might be worth adding a small graphic to signal that their content is incoming.
- **Modals** – If you want to provide contextual messages for users as they navigate, you must rethink modals. The traditional pop-up style modal will only annoy users. Instead, embed the modal as part of the scroll like we show in this animated tutorial.

- **Haptic feedback** – Small vibrations work well for accompanying alert messages. To learn more, designer Sally Jenkinson explains how to implement vibrations.

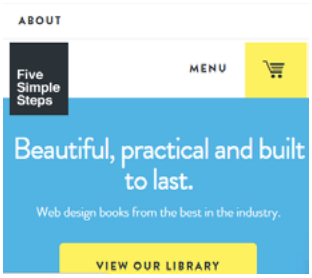
Consistency

Different screen sizes and device form factors require different kinds of website navigation. It's a mistake to insist on absolute consistency across the different layouts of a responsive site.

Here's a good example of how to modify navigation for responsive design:



Starting at the a tablet-sized viewport, FiveSimpleSteps uses a horizontal navigation.



As you shrink to the smallest viewport, the navigation changes to simple “MENU” button which expands above when revealed.
When designing responsive navigation, the following items should remain consistent across all viewports:

- Link or button labels
- Typefaces
- Color treatments

The following elements should not remain consistent as they accommodate the nuances of specific screen sizes:

- Button size
- Visual layout
- How the navigation works, including the differences required for touch screens. For example, a desktop navigation can sit static at the top of a screen. When you rework the layout for the mobile screen, you can make the navigation persistent and shrink in size to get out of the way.

Finally, to preserve the consistency of the experience, try not to link to sites that aren’t mobile friendly. You might easily overlook this detail since you don’t own the designs, but remember the the experience is continuous for the user.

Clarity

So far, everything we've discussed speaks to a required sense of clarity in your responsive navigation. Now let's explore a few direct tactics.

1. Clear Labels

   Specific labeling makes a huge difference in this regard. Menu options shouldn't be ambiguous on any level. If you want to know more about the perks of a product or service, think about which you're more likely to click:

   - a "Features" page
   - a page labeled: "the cherry on top."

   While the second option is cheeky, it doesn't precisely describe what's actually on the page. If you want to know why the product or service will be an asset to you in your daily life, then you'll be looking for the benefit, not the idiom used to signify the benefit.

Photo credit: MailChimp

   Known as "Information Scent", you must create labels to help users get on track to the desired content.

2. Search As a Backup Option

   A search option is important, if not an outright requirement.

   - Conceptually, search navigation lets users create their own navigation.

- Practically, it means you don't have to make information-dense navigation panels or menus.
- Technically, it does require more work up front, but the long-term results are worth the effort.
- Structurally, it cuts across information architecture, so be sure that any results are meaningful and complete.

It's essentially a shortcut or a navigation hack for your users. You can't predict every path they'll choose to take, so you must allow for a "choose your own adventure" option. As we touched on before, the context users bring to the table is often unknowable, so it's only wise to offer a shortcut.

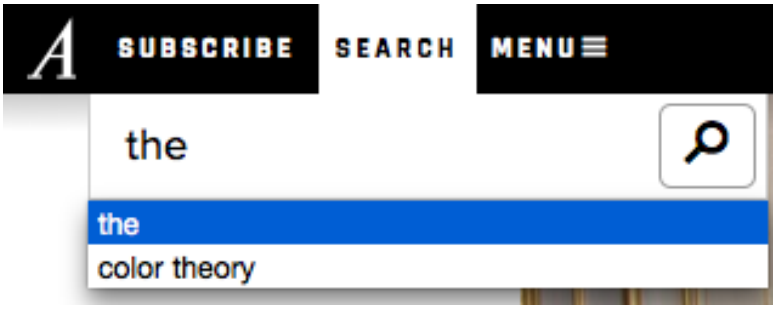


Photo credit: The Atlantic

Even so, your primary navigation controls should take as much of the user's context as you can glean into account. Do not rely on search as a crutch for bad navigation.

What destinations are most likely to be their preferred paths? Don't be satisfied with just listing your site's top level sections. Use your research into their personas as a jumping point for which pages will be most prominent in your navigation controls.

3. Alternatives to the Hamburger Menu
   Icons may confuse some users.

   We aren't saying to avoid the hamburger menu at all costs. Rather, just make sure you understand that the hamburger icon isn't as usable as many designers seem to think.

   For instance, if you're designing for less tech-savvy users, they may lack familiarity with UI patterns.

   Either way, you're usually better served enclosing the icon in a button to indicate that it's clickable, or even replacing it altogether with text that reads: "MENU." The idea here being that you want all users to understand the location of navigation options, not just the ones familiar with UI trends.

Photo credit: Smashing Magazine

You could even combine both the MENU label and hamburger icon into the hybrid pattern below.

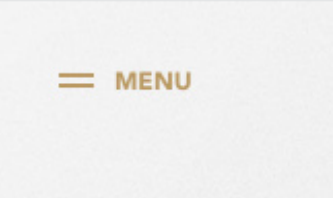


Photo credit: Haute Horlogerie

4. Navigation As Content

   And finally, this is your chance to actually *design* navigation as content.

   Who says navigation must be a list of links? Why can't it be a block of teasers with art, a timeline, a carousel, or act as a geographical map?

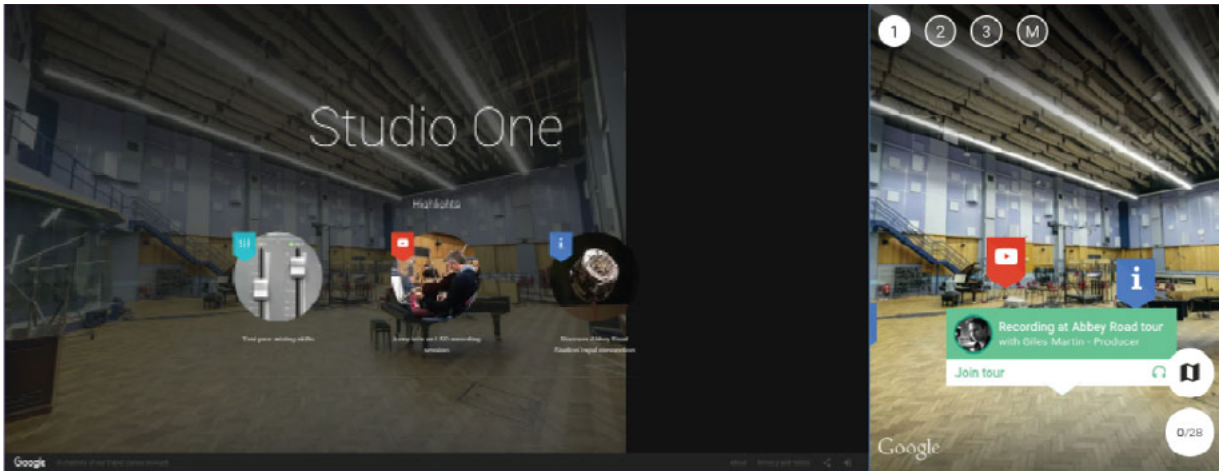


Photo credit: Inside Abbey Road

   Be creative and have fun. Just keep an eye on usability to ensure you don't go too far off the rails.

Useful Navigation Patterns

As Brad Frost shows in this extremely informative article, many useful responsive navigation patterns exist.

For the sake of brevity, we'll focus on 3 of the most popular types of effective navigation. And with our final case study in this section, we'll give you a more solid example of what it means to design navigation as content.

1. The shrinking sticky bar:
   the navigation menu stays fixed to the top of the screen. As the size of the viewport decreases, it centers and shrinks in response.

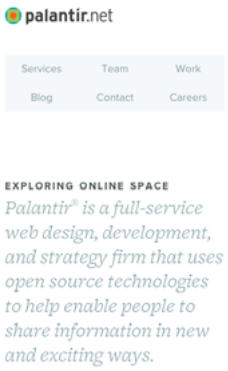
   This method is common mostly because of the relative ease involved. The pattern is often referred to as the "Do Nothing" approach. It's a decent start, but it might not be as scalable in the long-term. As your site content grows (e.g. adding in "Services" with "Products"), you have no choice but to eat up screen real estate with the navigation.

   Your navigation options are prominently featured, which is obviously a good thing. But unfortunately it can come at the cost of content.

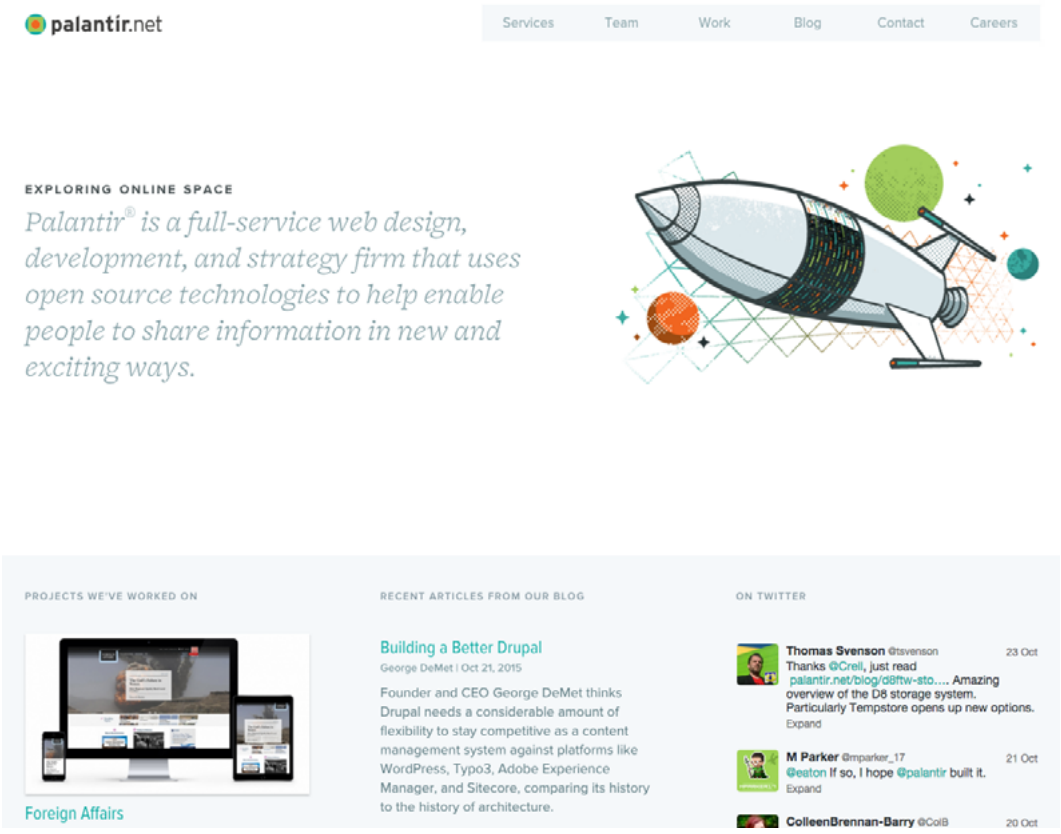
Case study: Palantir

For Palantir, notice how the “Do-Nothing” navigation is already pushing its limits. While it currently functions just fine, another navigation item would disrupt the two-row structure.

Smallest Viewport:



Largest Viewport:



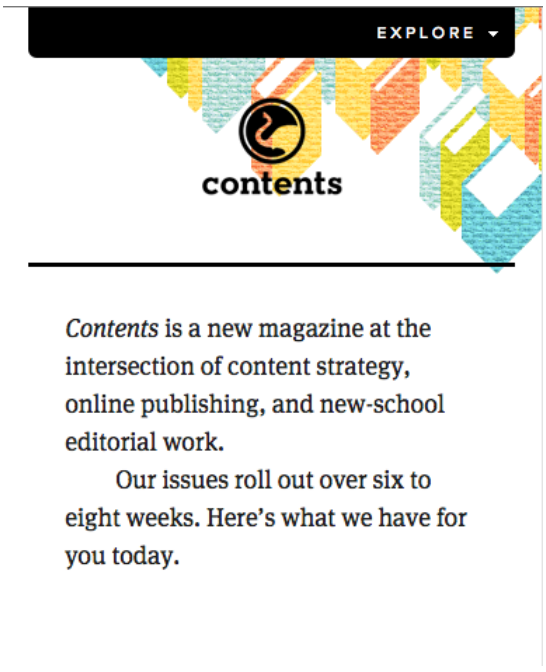
2. Anchor to the header:
   a fixed-position button that links to the header.

   In this pattern, the navigation resides in a large footer at the end of every page. The button is a small, ever-present link that takes users to that header. In our case study example, it takes the shape of a “Back to the top” button.

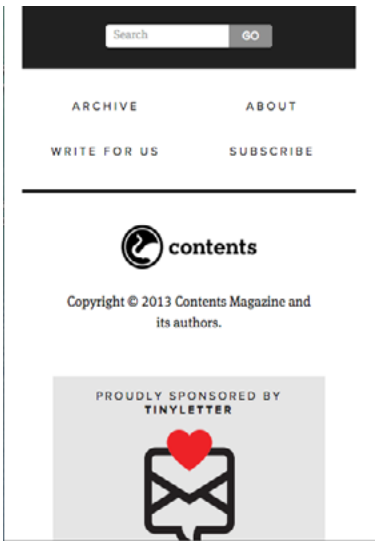
   While this format is more scalable than the “Do Nothing” approach since you just expand the footer to accommodate more items, the transition to the bottom of the page can feel a bit jarring. You do, however, save space in the viewing window since the navigation is only a simple button.

   Case study: Contents Magazine

   Smallest Viewport (top navigation):



Smallest Viewport (footer navigation after clicking “Explore” in top nav):



Largest Viewport:

3. Off-canvas navigation drawer:
   tap an icon to reveal a “hidden” menu from just out of sight.

   This pattern often takes over the entire screen, and has a “close” button in case the user changes his/her mind about wanting to navigate away from the page. Most commonly, these drawers appear to slide in from the left or right.

You can design off-canvas drawers if they are an entire page by themselves, in which case they typically appear just as a list of links.

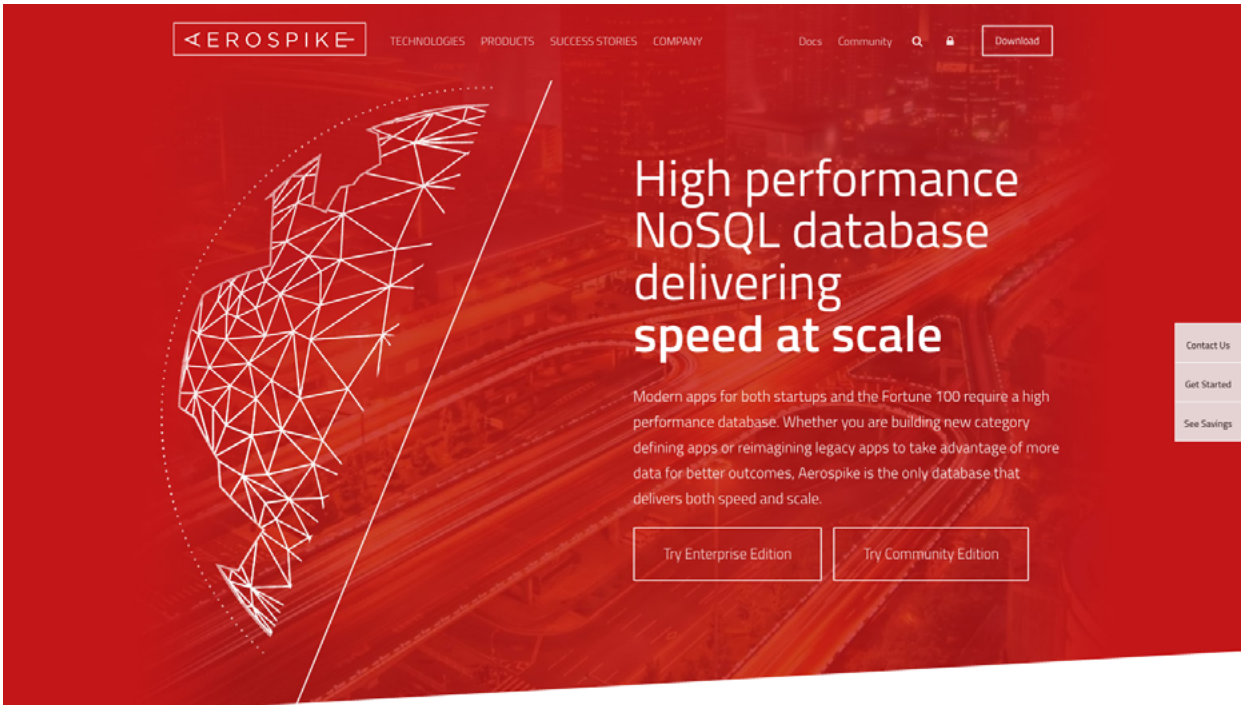
Off-canvas navigation is quite scalable since the navigation hides in a drawer, meaning other on-screen elements aren't affected by adding more items. However, because the menu is hidden, you must clarify the navigation button. For example, a "MENU" button works quite well.

Our two examples show a simple off-canvas drawer that could almost be described as a massive dropdown menu (Aerospike), and then a standalone navigation page (Uber).

Case study: Aerospike

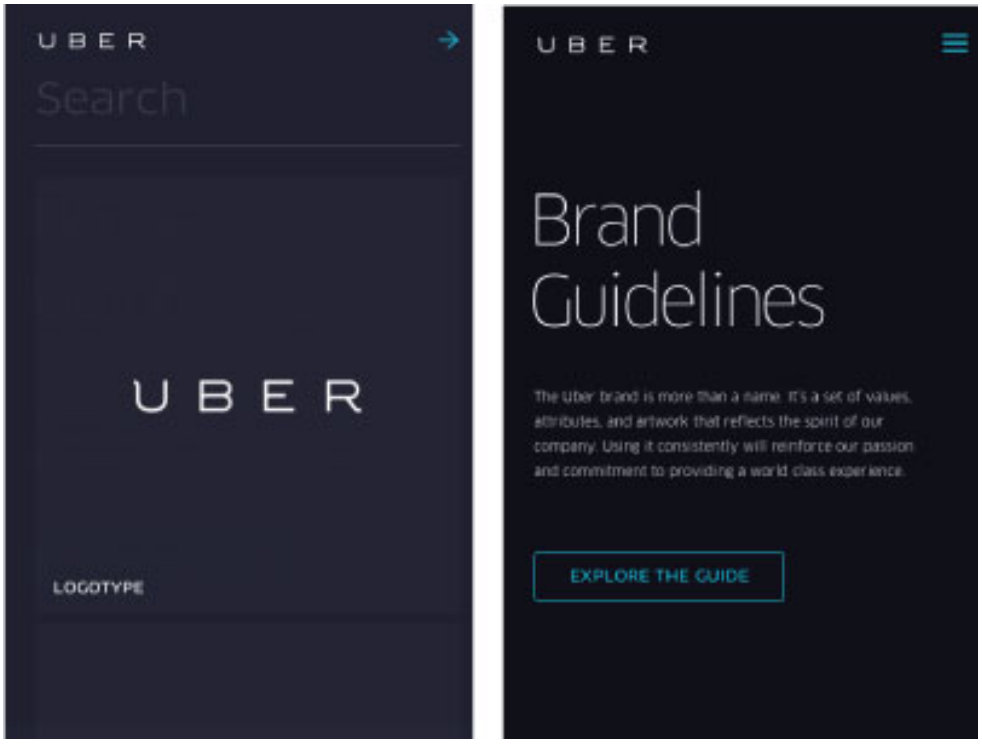
Smallest Viewport:

Largest Viewport:

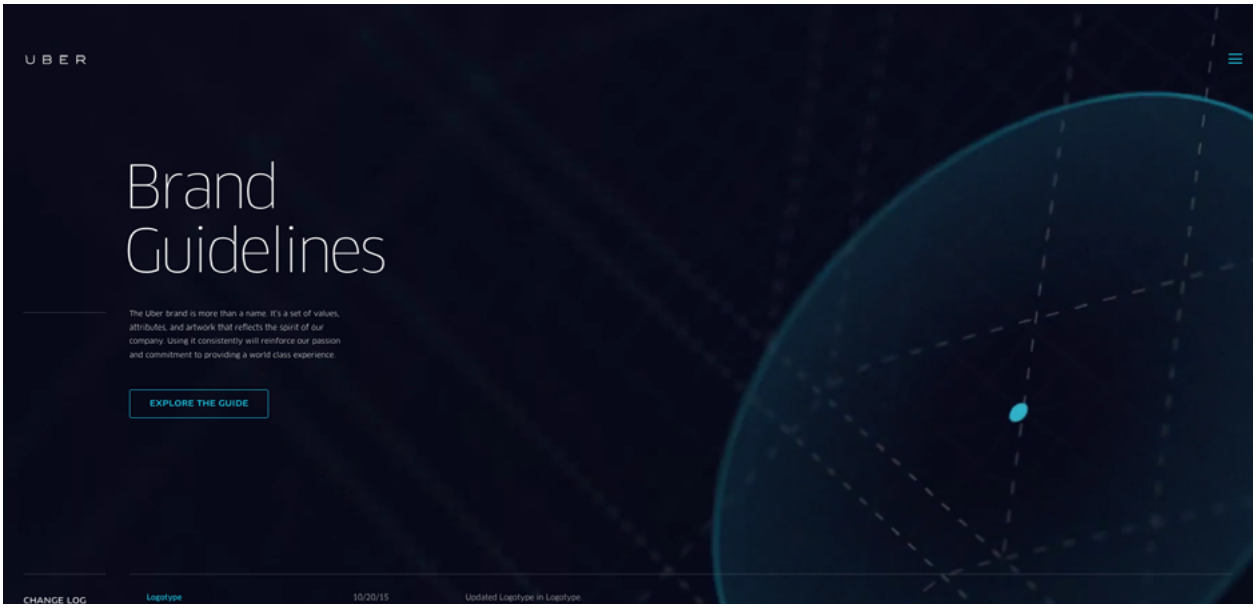


Case study: Uber brand guidelines

Smallest Viewport:



Largest Viewport:



4. Navigation as content:

   Let the users define their own experience while making simple links more like teasers.

   This pattern relies on the narrative of the content instead of a traditional navigation menu. As a result, users will scroll more

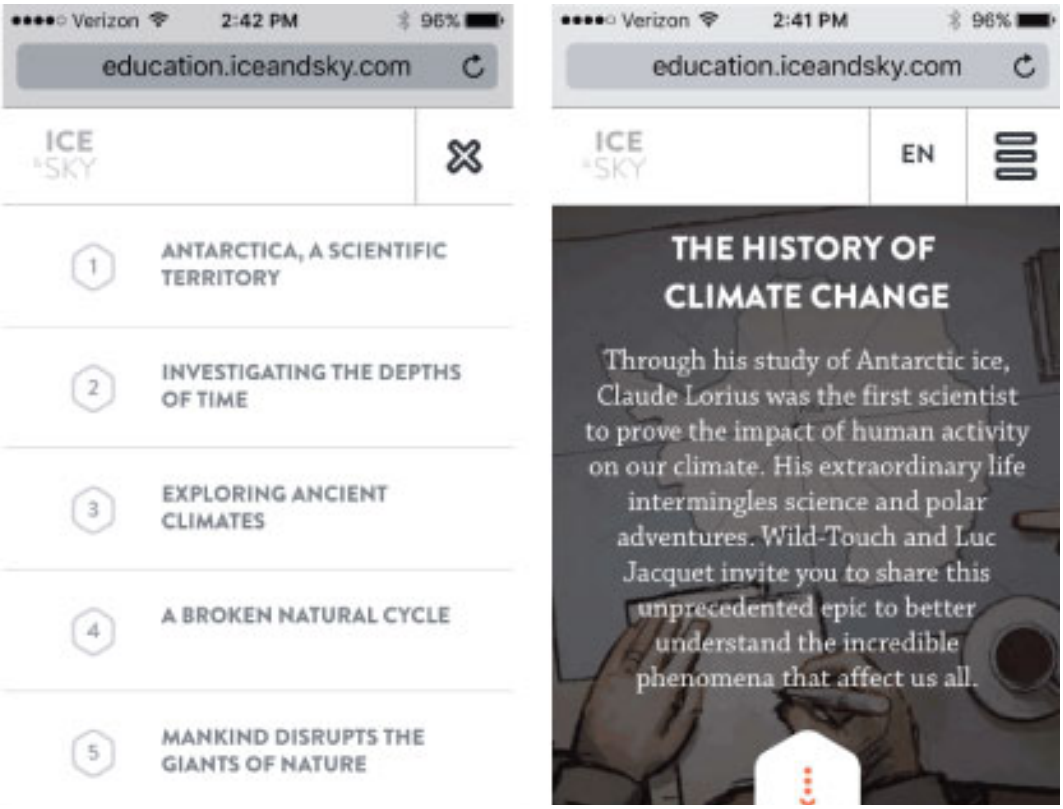
than they click. If your site lends itself to a linear narrative, this pattern is extremely effective.

The case study below exemplifies what's possible with creativity in navigation. Designed to navigate a story rather than a web offering, take note and feel inspired for how you can design your navigation options as content in and of itself.

Case study: Ice and Sky

As this site shows, the "navigation as content" pattern also pairs well with a minimalistic drawer. The combination allows users to scroll through a storytelling experience while still retaining control by tapping to specific sections.

Smallest Viewport:



Largest Viewport:

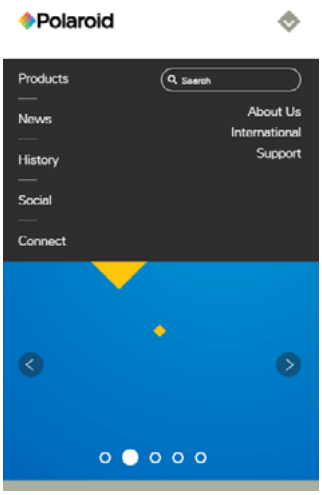
Additional Examples

Of course, there are more than 4 ways to skin this particular cat. Here's a gallery of a few other common navigation patterns:

1. Polaroid – off-canvas navigation from the top

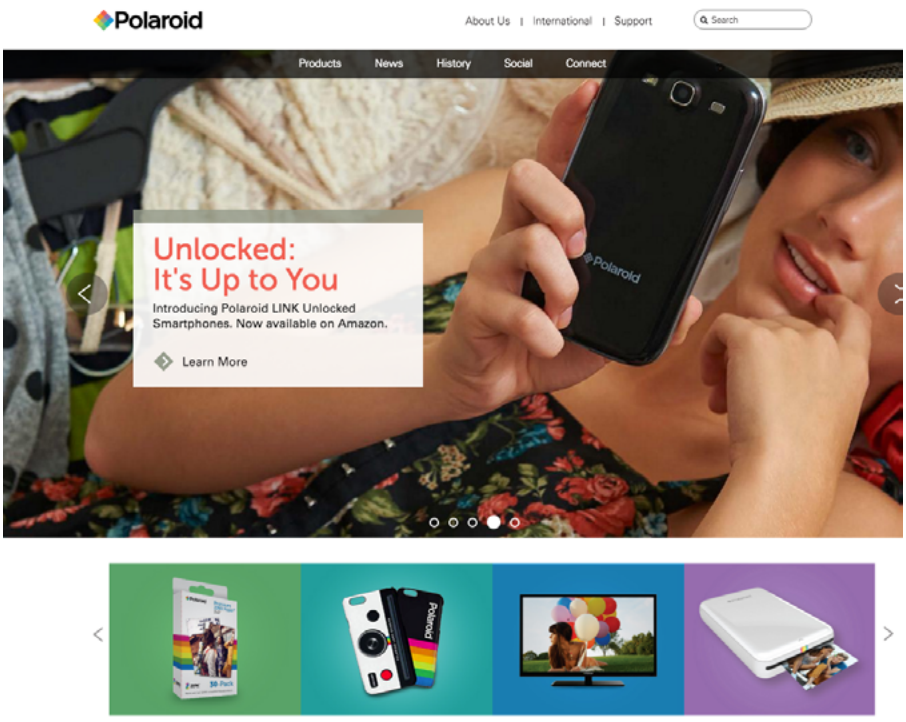
   Smallest Viewport:

   In the smartphone view, all of your options are back in a handy top-down toggle menu.



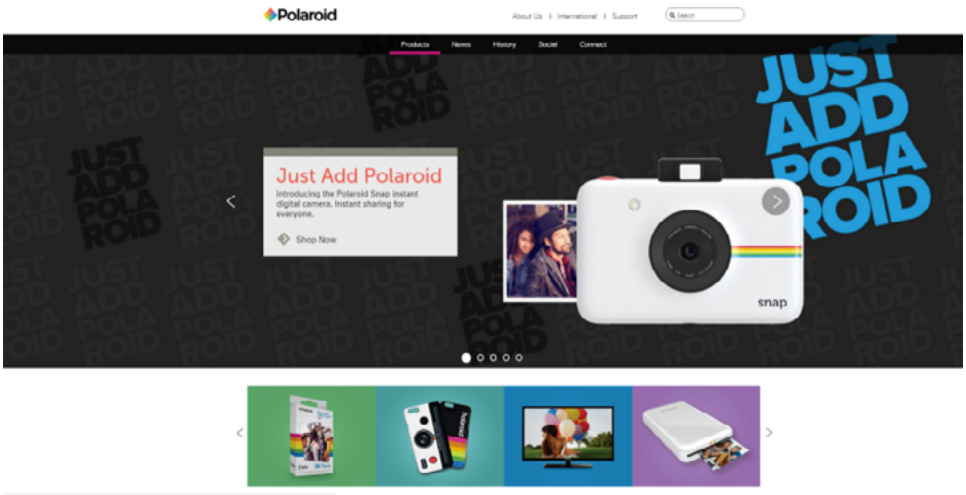
Mid-Sized Viewport:

The tablet view expands the navigation horizontally and adds a primary CTA in the middle of the screen along with 4 secondary CTAs at the bottom of the screen.



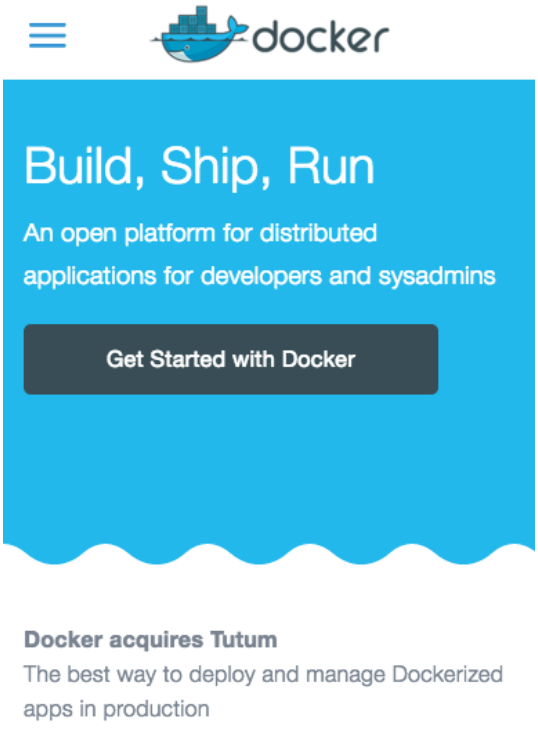
Largest Viewport:

The navigation and content layout remain the same as the tablet-sized viewport.



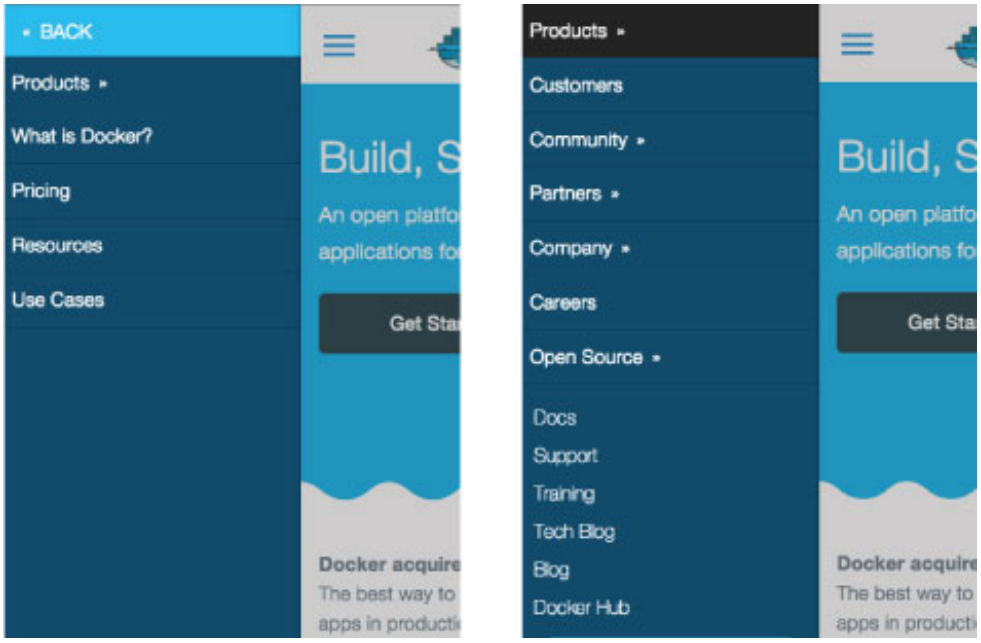
2. Docker – multi-level off-canvas navigation

Smartphone View (menu inactive):



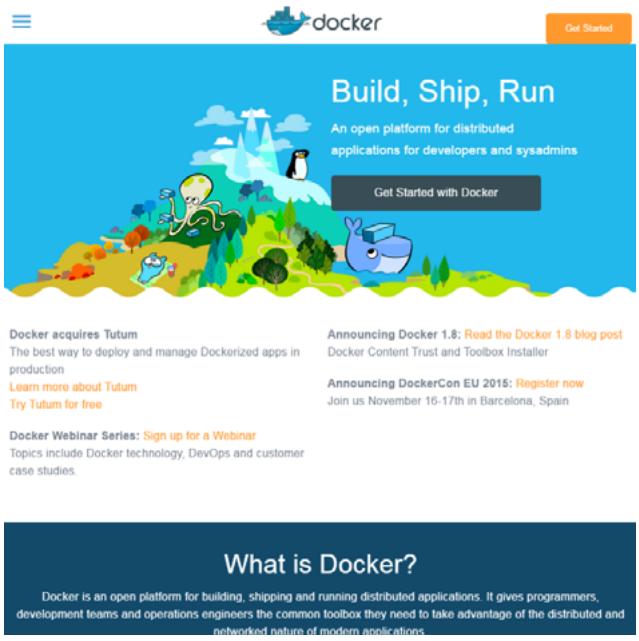
Smartphone View (menu activated, menu expanded):

As you can see the content is dimmed but visible, while the navigation options take center stage:



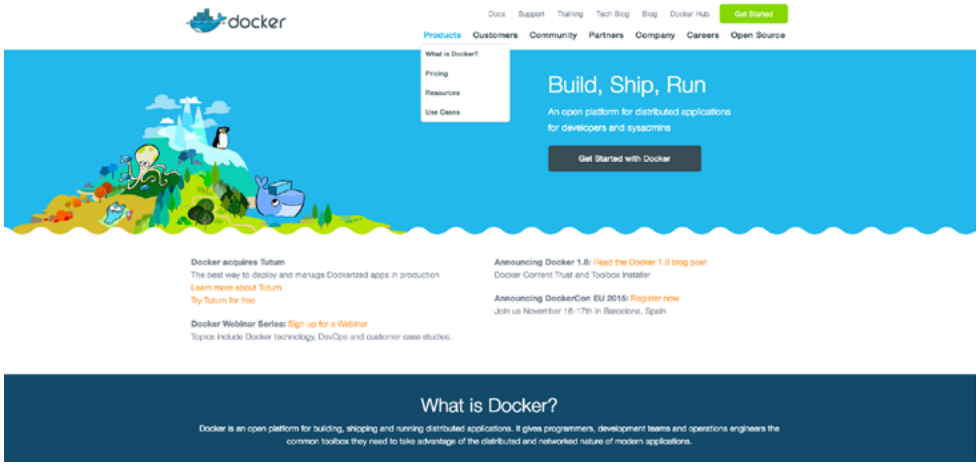
Tablet view (note the hamburger icon):

At this point, the menu moves from the top to the side and pops out from the left when the icon is clicked:



Largest View:

Like we've seen with other sites, the top navigation now spreads out horizontally. In this case, the multi-level drawer menus become dropdown menus.



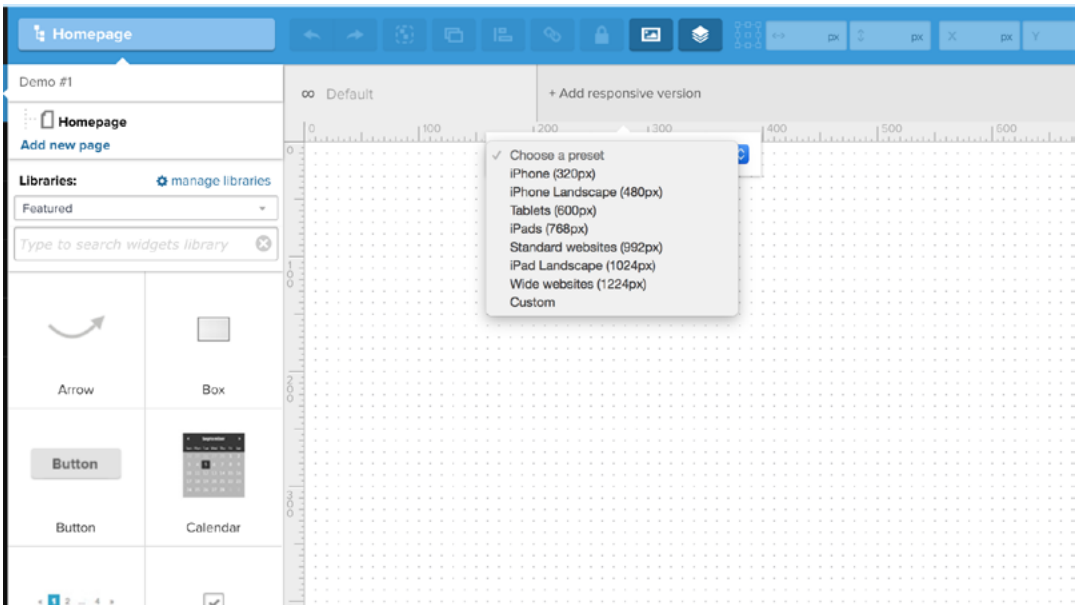
Tutorial: Responsive Navigation Drawer

Now that you know the best practices, let's apply them by building out one of the patterns we discussed.

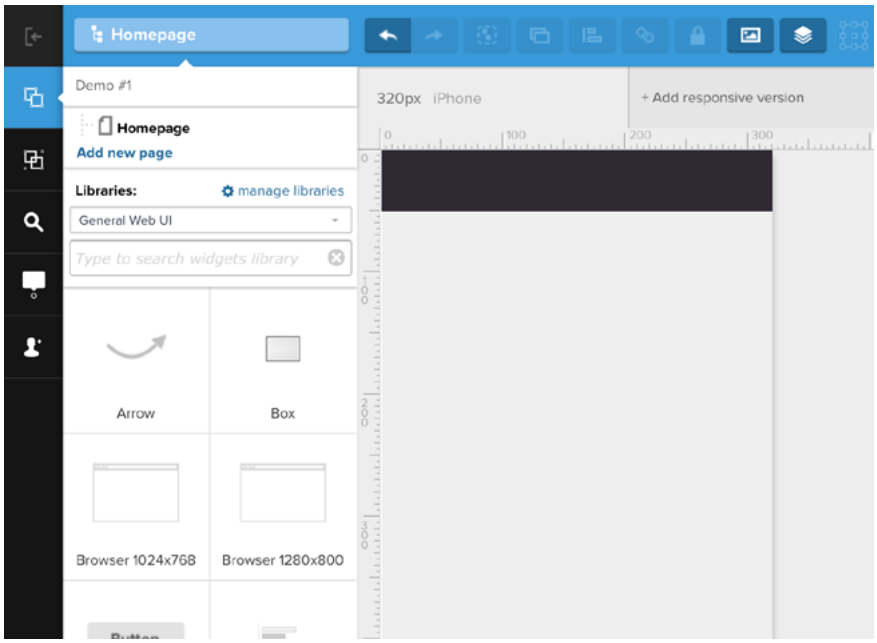
We'll create an off-canvas navigation drawer for the smallest viewport (usually set for smartphones). Once you're done, you can adapt the navigation for the mid-size and largest viewports by laying out the items in a horizontal menu (as shown before).

Prototyping the drawer used to require certain code skills, but with the latest addition of advanced animation editor, UXPin made it as easy as 1-2-3. It takes no more than 10 minutes to create a full-blown navigation drawer. Let me walk you through the process step by step.

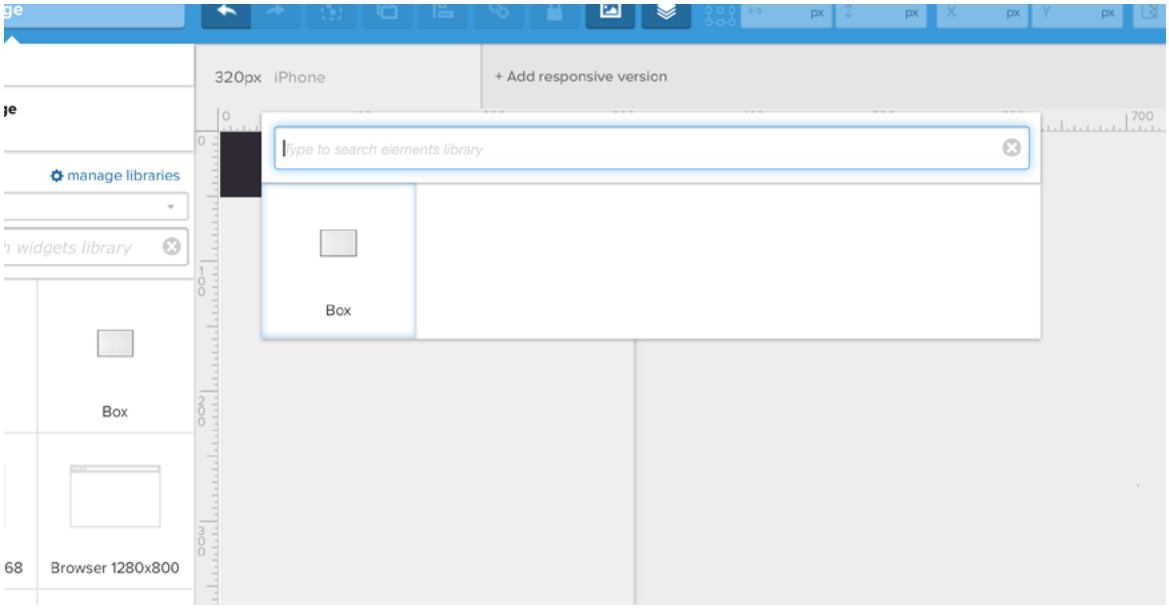
1. First, we have to specify the width of the canvas. We'll do it with the responsive breakpoint. If you're designing for the iPhone 5s then choose 320px.



2. With the canvas ready, it's time to start designing. Add the top bar and background using "Box" elements. The box is available in "General Web UI library." Using variety of UXPin's libraries you can quickly build whatever you want.

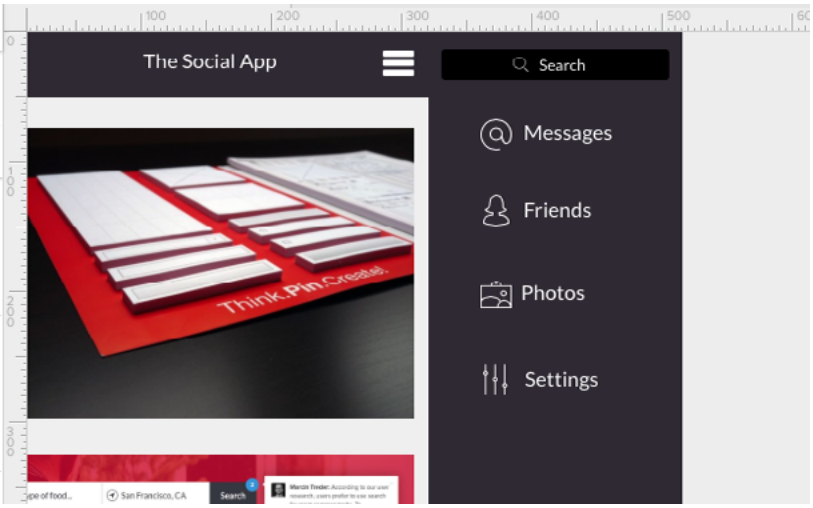


For a quick navigation, use “search” (shortcut: cmd+f on Macs, ctrl+f on PCs).

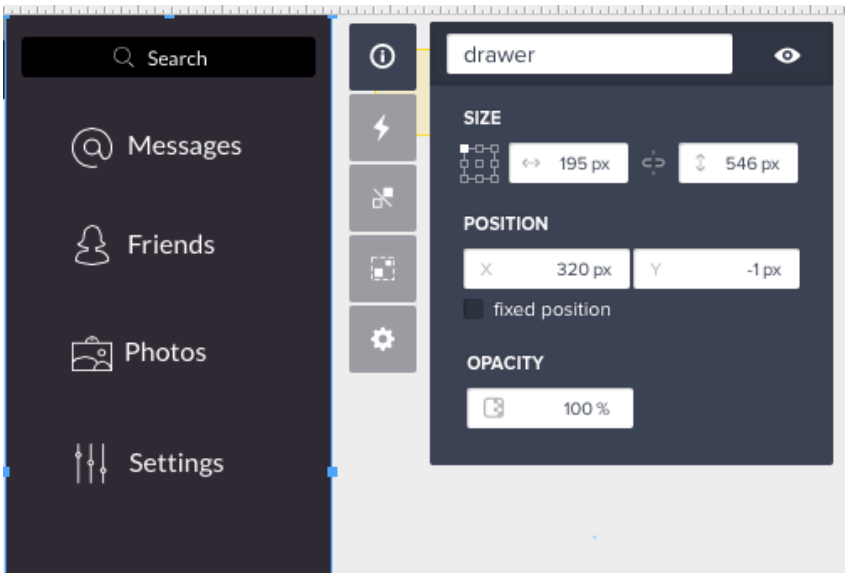


3. Don’t forget to add the infamous “hamburger icon” or any other symbol representing menu (use search to browse all the icons libraries in UXPin).

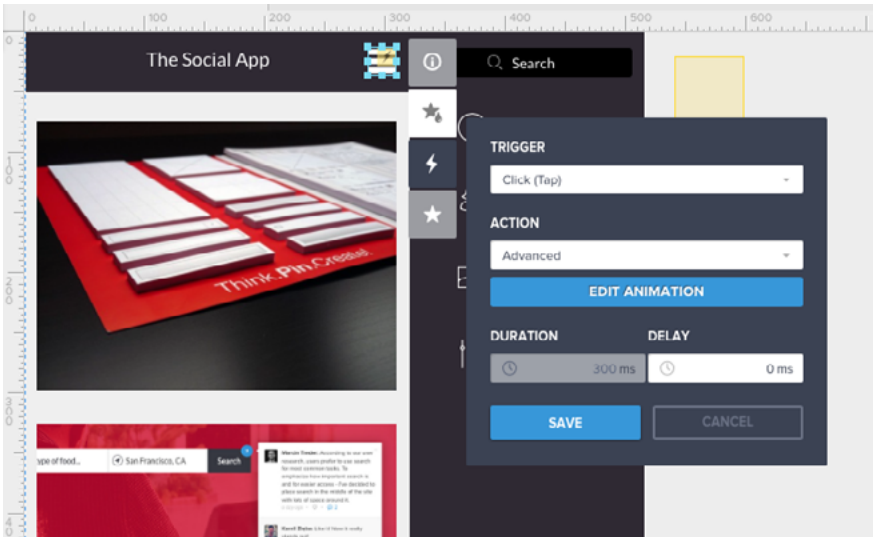
4. When your main screen is ready, build the drawer menu. Place it outside of the canvas. To start with, it should touch the right edge of your main screen.



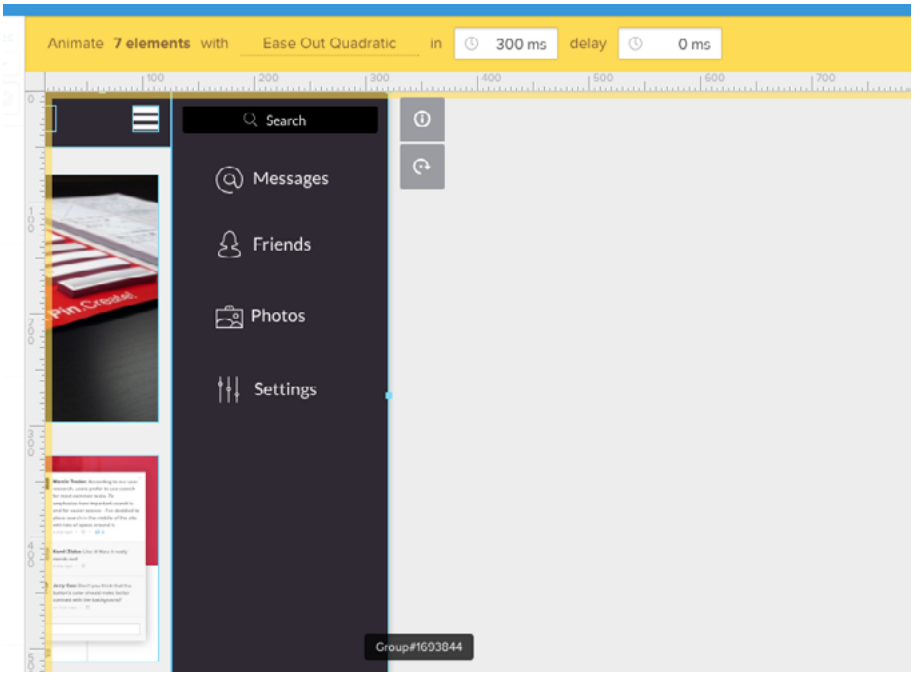
5. Select every element constituting the menu and group them together (cmd+g on Macs, ctrl+g on PCs; or button in the top menu). For easy future reference, change the group's name "Drawer." Depending on your design of the "main screen" you might also want to group to help with the fluidity of the animation.



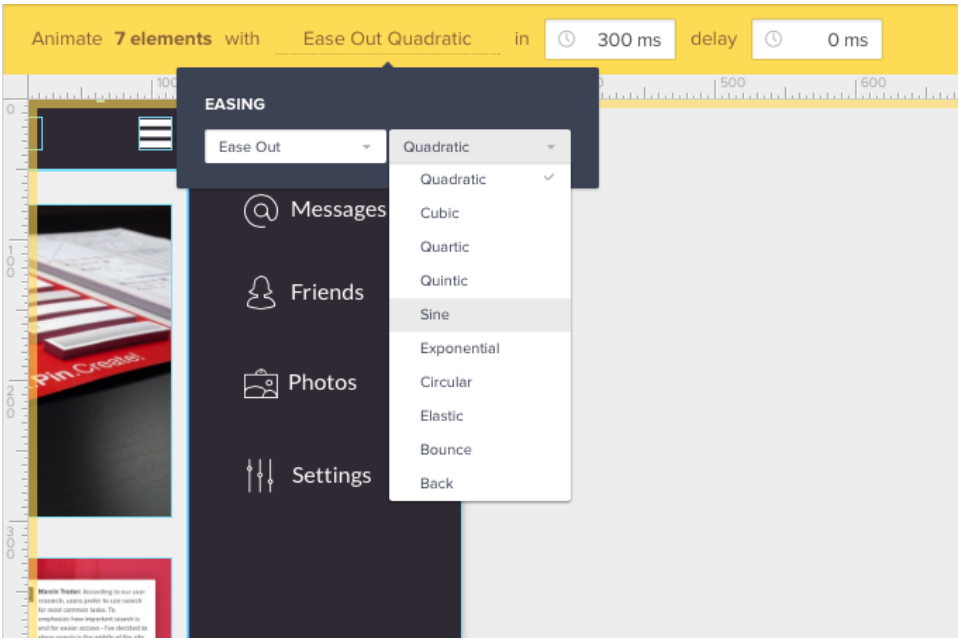
6. Time for interactions. Left-click on the hamburger icon and from the properties manager menu choose "interactions" with the "click (tap)" trigger and "add advanced animation" instead of "action." That's your first step to great navigation drawer.



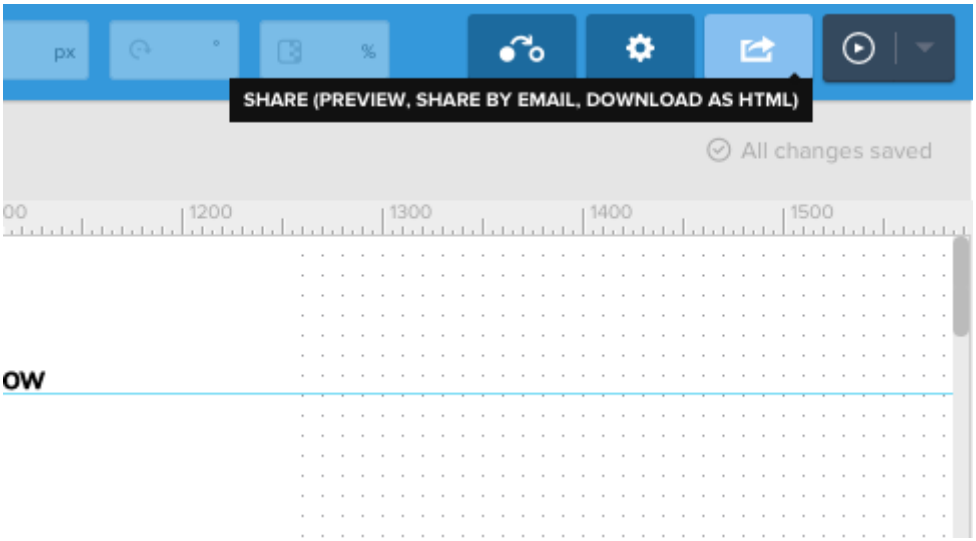
7. In **the advanced animations editor,** select all of the elements and drag them to the left. The far right edge of the menu should touch the edge of the 320px-width canvas.



8. We're almost there. Time to make things smoother. In the top menu choose the kind of animation you would like to use. I'm going to use "Elastic" with "Ease In."



9. You got it. Just click “save” and simulate your design using UXPin Preview or send it directly to your mobile phone.

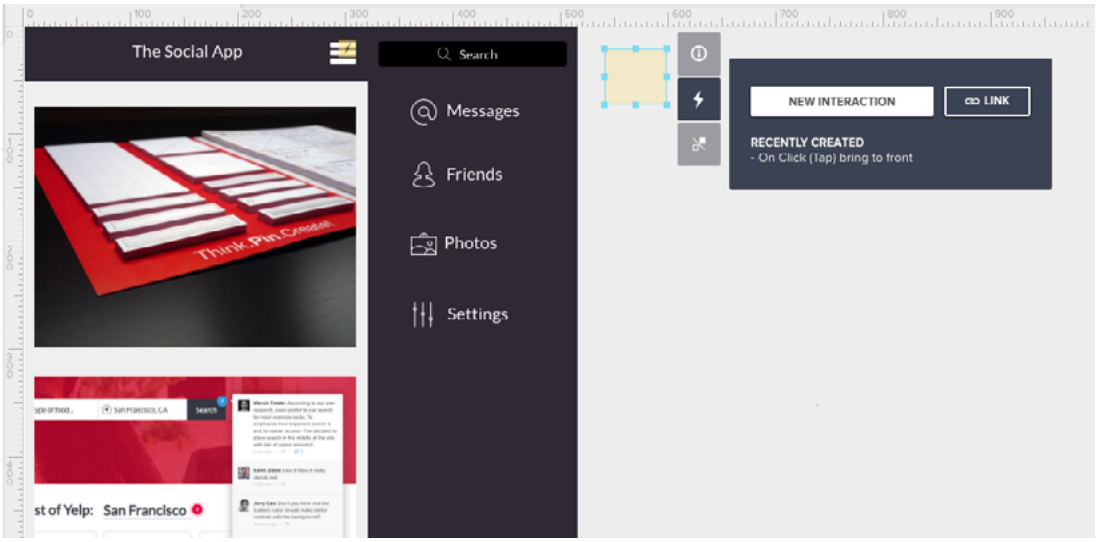


It’s ready!

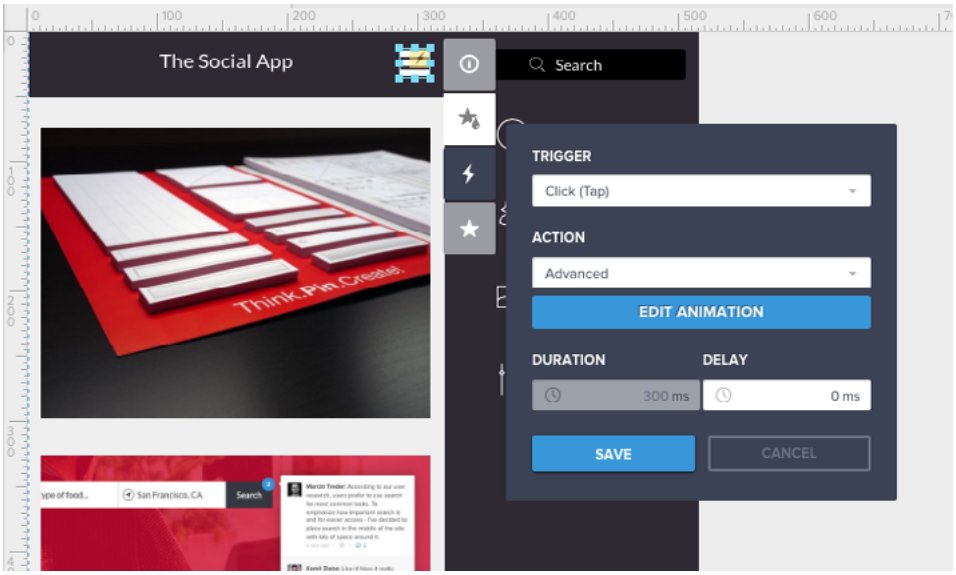
Do you want to make it even better? Take a look at some advanced options for the navigation drawer.

How to add “a come back”

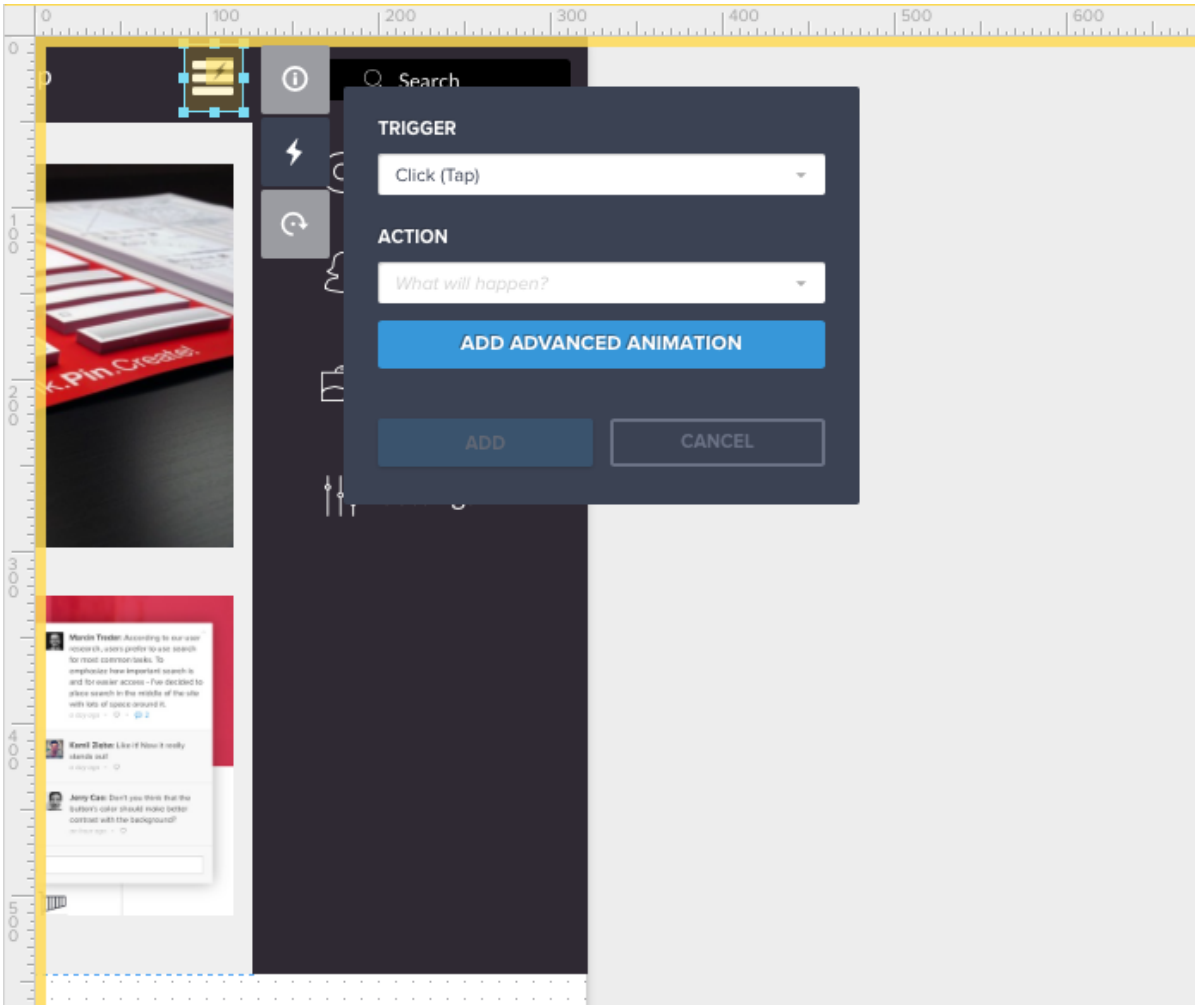
1. We have to start with something slightly counter-intuitive. Bear with me. Since each element can only have one advanced animation with the same trigger assigned, we need to drop “a hot spot” (element available in General Web UI library) anywhere outside of the canvas.



2. Now click on your hamburger icon and enter into edit mode of your advanced animation.



3. While in “Advanced Animations Editor” move your hotspot directly above “the hamburger icon” and add Advanced Animation to it.



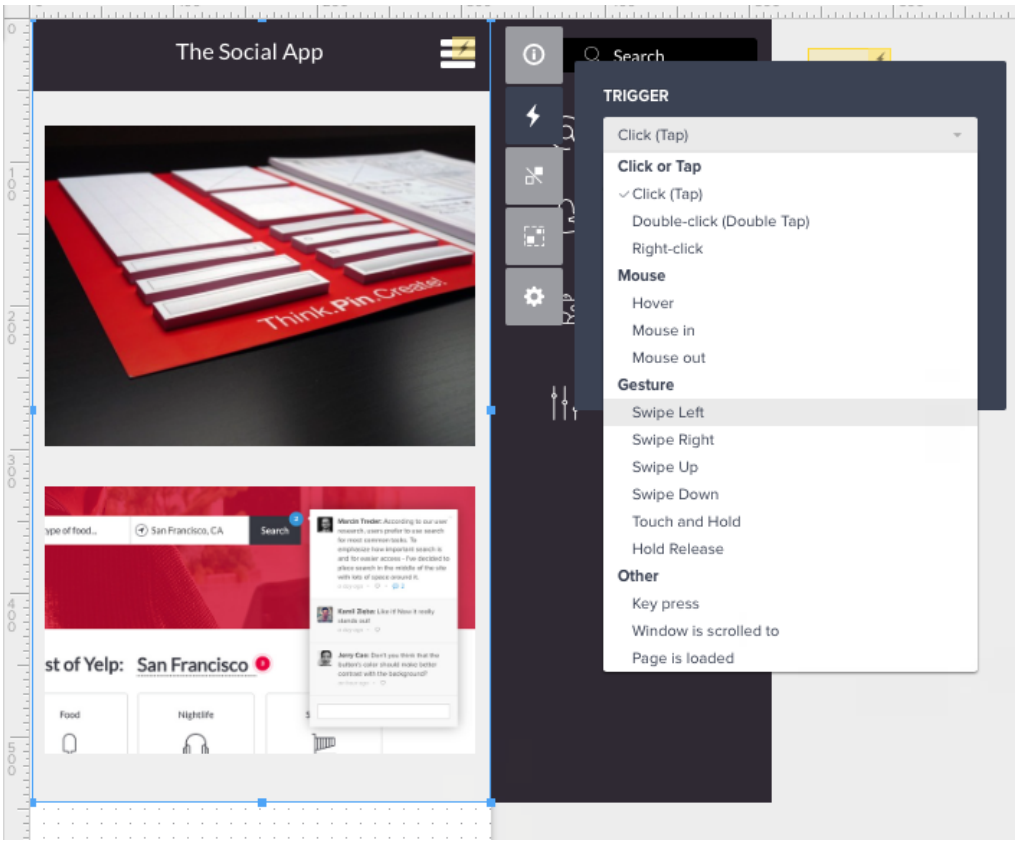
4. You just entered advanced animation within advanced animation. Sounds scary, right? Have no fear. UXPin will automatically add a “come back” as the second step in the animation, so just click “save.”

5. Might be hard to believe, but you’re done! Check your design in UXPin preview. Smooth, huh?

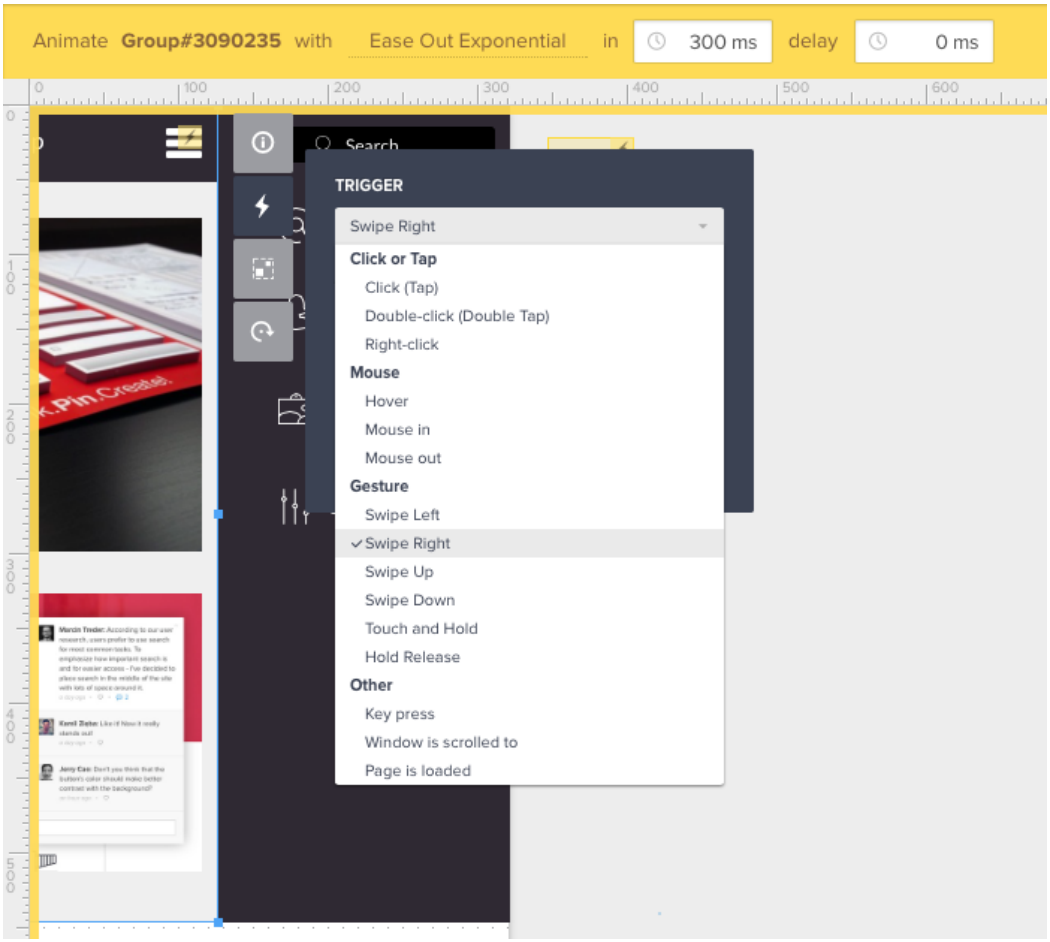
How to show the navigation drawer on swipe to the left and hide it on swipe to the right

Ah, swipes! They used to be impossible for prototyping tools. Not for UXPin though! Let me walk you through the process.

1. Click on the main content group and add interaction: trigger “on swipe left” with action: “advanced animation.”



2. In advanced animation editor, move all of the content to the left. The right edge of the navigation drawer should touch the right edge of the canvas. Click on the main content group and add interaction: trigger “on swipe right,” action “advanced animation.” UXPin will automatically generate a “come back” to the previous state. Click “save.”

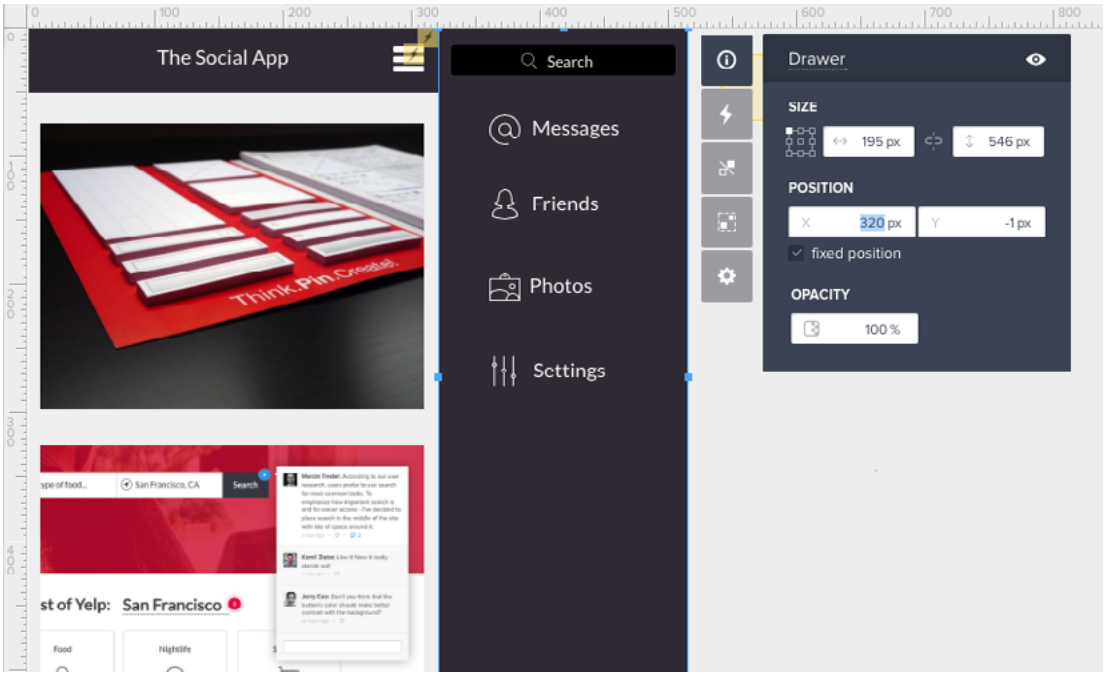


3. Done! Test it on the UXPin preview, or send it to your mobile phone.

How to make “left-side” navigation drawer

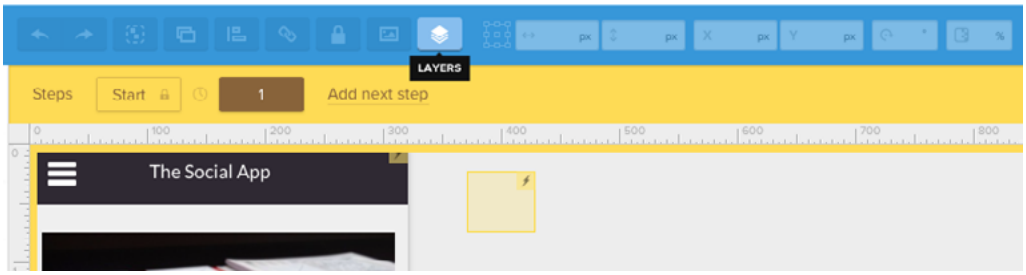
Time for the real troublemaker. Let’s create “left-side” navigation drawer.

1. Click on the drawer navigation and in its properties set horizontal position (x) to –200. It will disappear from the screen, but don’t worry! Trust me, everything will be fine.

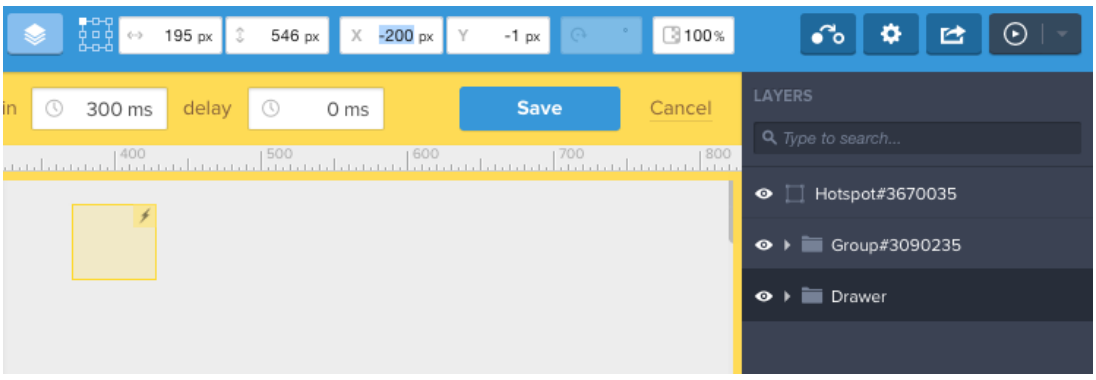


2. Add an advanced animation to the hamburger icon.

3. In the advanced animation editor, open “layers.”



4. Select “Drawer” from the list of layers and change its horizontal position (you can use the top bar) to 0px.



5. Now, just move the main content group to the left and click “save.”

6. To add a come back on the left-swipe, edit the advanced animation attached to the main screen and add swipe-left animation while inside of the animations editor.

7. Done! Test it on UXPin preview, or send it to your mobile phone.

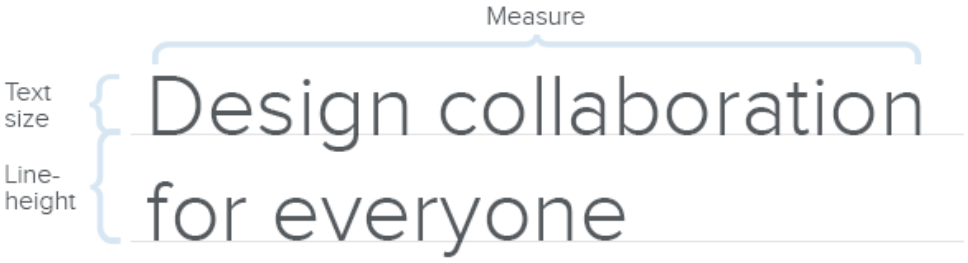
Hope you enjoyed this tutorial. If you’d like to build your own responsive prototype, go ahead and get started in UXPin with a free trial.

Responsive Typography

The written word is a top consideration on any website. It's the way we communicate actions, goals, functions and offerings to the user. So it's no surprise that legibility, readability, and overall presentation all have a major impact on the ways that users see and understand your designs.

A few important initial points on responsive typography:

- Responsive typography reinforces the authenticity of a design, making it more credible since the text fits the device naturally.
- In general, typography defines the structure, behavior, and presentation of your content. If content determines design, then typography illuminates the content. So you have to be sure that this base level design is acceptable across different screen sizes, otherwise the overall integrity of your design begins to break down.
- Line height, text size, and measure must all correspond proportionately to the size of the screen. This requires the use of relative units of measure (ems and rems).



First, typography must be presented correctly at every screen size in order to remain legible and attractive. Unfortunately, this can have less to do with screen size and more to do with the user's distance from the screen. While many designers pick fonts based on personal preference, the true essence of responsive design requires a closer look at user context.

Reading Distance

Text size hinges upon reading distance: how far the user's eyes are from the screen.

A user may have a laptop hooked up to a 64 inch big screen television to view your page, or they may have an iPhone 5s (124 mm). In one case, the screen is across the room and in the other it's maybe a few inches from the user's face. But your typography must still be readable in both situations.

The requirement can obviously put you in a tough position when determining text size at different breakpoints. How can you be sure how far a reader is holding their device? Short of technology that detects the user's distance from the screen (and we do have top designers

working on that) you'll have to follow the best practices suggested by responsive design research.

Keep in mind the following when accounting for reading distance in your designs:

- The further the distance between user and screen, the larger the text should be.
- Compare your website with print material at similar distances to determine what's readable. Desktop material should look much larger than printed material when compared side by side.
- Increasing text size can give a designer fits, but don't be discouraged. It's normal to feel agitated at this sudden change. Once you're used to the larger size, however, you won't be able to go smaller in good conscience.
- Desktops are typically further from users than handheld devices, meaning their font size should be slightly larger.
- In terms of reading distance, tablets tend to fall between smartphones and desktops–users are commonly holding them a small distance farther from their faces than would be the case with smartphones.

Ideally, your text at every breakpoint should look like it's the same size when held at reading distances on different devices.

Establishing Responsive Text Size

Your font choices will undoubtedly affect the base sizing of your text. However, 16pt body type is both the default on most browsers and a good place to start for all screen sizes. Even so, it's not the end-all requirement. There's a lot of room for experimentation, so feel free to make small adjustments using 16pt as a baseline. For example, since smartphones are held closer to our faces than desktops, we can reduce the body type to 14 points (but 16 also works).

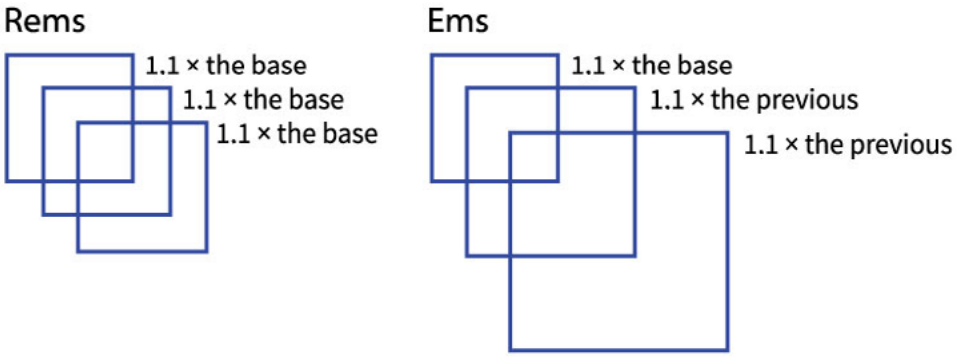
When dealing with headings, on the other hand, it's very important to clearly differentiate them from the body text. For all browsers, make sure that heading text is at least 1.6 times as large as the text supported by the headings.

The most important consideration with responsive text is, of course, that it changes to fit the context in which it's being viewed. To do this, it's highly recommended that you set text size in relative rather than absolute units. In other words, use ems or rems instead of pixels. Even if you don't own your own code, ems and rems will still help you approach typography in terms of proportions (which is the foundation of all responsive design).

- **Reference Pixels** – The smallest unit of measure for area on a display screen. They are an absolute unit of measure rendering the same way on any screen.

- **Ems** – A relative unit of measure which determines font size in a defined proportion to its parent element's font size. In a way, ems increase font size exponentially.
- **Rems** – Another relative unit of measure which determines font size in a defined proportion, not to the parent element but instead the root <html> element's font size–regardless of how deeply they're nested.

Since ems compound on each other, coding your font size with ems will require caution. Adding too many ems will result in complex and unattractive code that doesn't render well in every situation. And you should be especially careful not to add ems to structural elements, keeping their use limited to typographic elements only.



Rems are easier to control than ems, and they're just as browser compatible. Because they don't have the same problems with cascading, rems can behave in the same ways as ems but with fewer complications.

In summary, we would recommend sticking with rems when implementing responsive typography.

Rules of thumb:

- Size text so it fits at between 22 and 60 rems wide. A 320px smartphone viewport at 14pt = 22.9 rems; 960px tablet viewport at 16pt = 60 rems. (Notice we're using points, not pixels. Multiply points by rems to get viewport size)
- Set base font to 100% of the browser default, rather than setting it to a number of pixels. Doing so makes your designs more accessible to user, allowing them to customize their reading experiences.
- Use rems whenever possible to simplify your work with relative units.
- For displaying links or calls to action, try more than one word to increase tappability. For example, instead of "Buy", use "Buy Now" or "Buy Item". To tie back into what we discussed, more than one word equates to roughly 5 rems.

Here's a quick chart that shows how to set text size in terms of points and rems. We'll use 14 points as the smallest size for the smartphone view. Remember that 1.6 rem is the *minimum* recommended for headlines. Your headline size may be larger depending on your site's visual style and hierarchy.

	Mobile View	**Tablet View**	**Desktop View**
Body Text	14 points	16 points	16 points
	1 rem	1 rem	1 rem
Headline	22 points	26 points	26 points
	1.6 rem	1.6 rem	1.6 rem

Feel free to set your font size and rems using this handy converter tool.

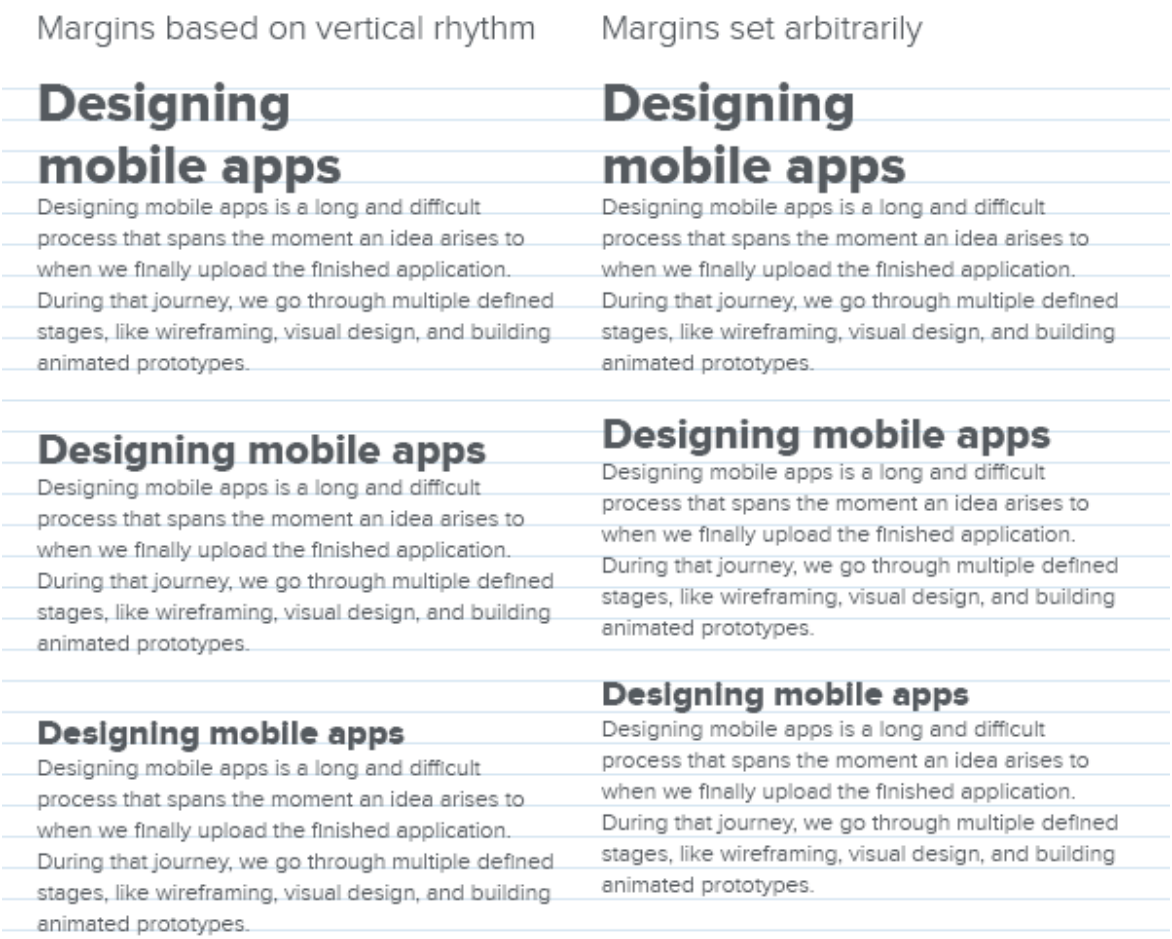
Line Height

As compared to text size, adjusting line height against screen sizes and reading distance requires a lot more trial and error. It will be up to you and your keen eye for design to make determinations on what looks good. There are however, some rules of thumb you can use as a good starting point.

In general:

- The greater the column width, the more line-height it needs to help users find the start of the next line.
- A line height of 140% (compared to text size) is a good place to start your tests, but this will vary somewhat based on your type-face. For instance, if your desktop view text is 16 points (1 rem), then your line height will be 140% of the text size. That equals 22.4 points (1.3 rems).
- Provided you've set the line height at 140%, you can keep your line height the same across all devices.
- Line height, like text, must remain fluid in order for your text to be responsive. Thus, it's a good idea to define your line heights with rems instead of pixels, just as you would normally do for your text. 1.4 rem is preferred for body text line height and 1.2 rem is a good size for your heading line height.

- Don’t forget to set appropriate margins relative to your body text line height between paragraphs in order to maintain the vertical rhythm of your text.



You can also set your line-height with unitless number values (in other words line-height: 1 rather than line-height: 1rem). This allows all descended elements to set their line heights relative to their own font sizes rather than those of the root element.

Line Length (Measure)

Longer lines of text end up causing reader fatigue. They also increase the chances of a reader reading the same line twice, which is a contributing factor of reader fatigue. For these reasons, it's advisable to keep your measure between 45 and 85 characters regardless of screen size.

While the ideal measure is traditionally seen as being between 45 and 75 characters, the wildly fluctuating screen sizes of the devices we use today have made this number a bit more flexible. Even so, as your screen width decrease, so too should your character count.

Here are a few other rules of thumb for character counts when using different layouts:

- For rows of text which aren't contained in columns, the ideal measure is 60 characters (applies to all viewports).
- For a two-column mobile view, keep the line length between 20-30 characters. When text is contained in multiple columns (on a desktop or tablet display) make line length between 45-60 characters.
- For mobile views, we wouldn't recommend exceeding 60 characters. For tablet and desktop views, a single column of text can allow a longer measure, between 70-75 characters.

If you need additional help with your line length choices, the Golden Ratio Typography Calculator is a helpful online tool that can automate your line length choices.

Conclusion

To sum up, keeping typography responsive requires careful consideration of user context.

What type of website are you designing for? Where is the user's face likely to be in relation to the screen? Use the answers to these questions to determine breakpoints and the accompanying text sizes. Remember too that line height and length must also be adjusted proportionally to maintain the text's legibility.

Finally, always use relative units, preferably rems, to define text size in relation to viewport.

If you can keep these lessons in mind and as always test to iron out inconsistencies, then you'll be well on your way to attractively rendering, always legible, readable, and completely fluid typography.

Responsive Images

Web design isn't cheap. It costs effort to produce and time to learn. And tools, like that fancy new Mac and Adobe's software subscription model, eat away at profits. But of all web design's costs to the designer, it's the *user's* costs we should consider first.

Mobile users often pay for every byte they download (and upload) away from wifi. HTML and CSS files, while getting larger every year, aren't as large as hefty JPG, PNGs and animated GIFs. Conscientious designers know that best practice includes making websites and apps that download as quickly as possible. It's like trimming out extra adverbs from copy, or extra div elements from markup.

If users don't need pixels, don't send them.

As we've seen, thinking "responsive" is more than slapping media queries into our code. Responsive images have their own set of challenges that designers must overcome. To make sites that work well and look great on a variety of screens, they need a smart strategy for images from the beginning.

Difficult? Perhaps. Worth the effort? Yes. A picture may be worth a thousand words, but if it weighs a million bytes, then users may give up before the picture downloads.

Choosing the Right Format for the Job

JPG, SVG, GIF and PNG (and PNG-24) – anyone new to web design may confuse the three. That's not surprising when even seasoned veterans opt for JPG when a SVG would do, or default for PNG-24 instead of PNG-8.

1. JPG

   Or JPEG, short for Joint Photographic Experts Group, was developed in 1991 and published in 1992 as a means to standardize pictures transmitted over the internet. Bandwidth was at a premium, so users prefered files that showed more picture for fewer bytes.

   The JPG format uses lossy compression, meaning that once applied, an image can never be fully decompressed back to the original quality. It trades smaller files for reduced quality on a scale of 0 – 100. Oddly, files with 100% JPG compression have the highest quality and the worst file size. 0% compression yields the smallest files with the worst quality.

100% JPG 0% JPG

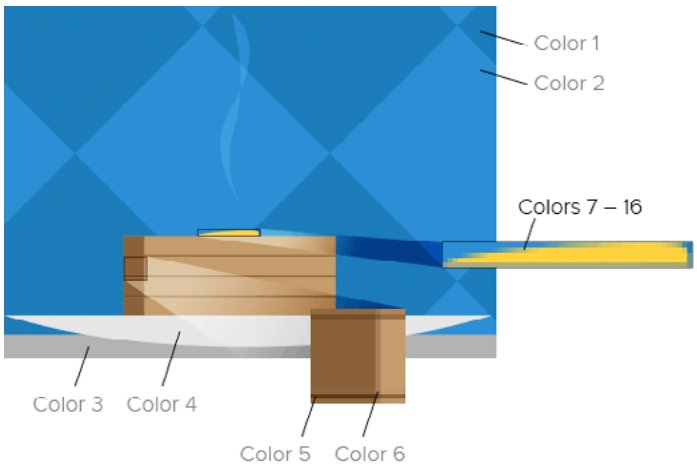
Artifacts are parts of an image that JPG compression changes for the sake of file size. They resemble blocks of homogenous color, when visible, as if herding colors into areas about 20 pixels square. Hard edges in images are the first victims of artifacts. That, and because JPG compression can look natural on complex images, means that this format is ideal for photos.

Bottom line: JPG compression works well for complicated images with lots of detail, like photos.

2. PNG (8-bit)

   Unlike JPG, Portable Network Graphics files use lossless compression that doesn't compound as the file is opened and resaved. Instead, PNG-8 files include a list of every unique color they use.

   And by unique, I mean *unique* as in #FFFFFF is not #FFFFFE, although to the unaided human eye they're indistinguishable. Each pixel is assigned to a color in the file's list, reducing the need for identical pixels to waste precious bytes reproducing what's already been said.



If the first hundred pixels use RGB(255,255,255), there's no need to say so – just state that pixels 1–500 belong to color #1. For this reason the PNG format is great at compressing images with perfectly flat colors.

The PNG-8 format can hold up to 256 unique colors in its list, called a *color table*. It can also make pixels fully transparent. These facts make PNG-8 ideal for today's trendy "flat color" look.

3. PNG (24-bit)

   Files that use PNG's other variety, PNG-24, look great because they use no compression. Nor do they use a color table. Every detail is preserved when saving PNG-24 files… and that's the problem.

   Opacity is another problem. While pixels in PNG-8's images can be transparent, it's all or nothing. Either you see them or you don't. Pixels in PNG-24 files can have *partial* opacity, meaning they tint elements behind them. Again, at the expense of file size.

When thinking responsively, PNG-24 is a last resort option. Only use them when you need fine transparency control and are willing to make users wait longer to download image files.

4. GIF

   Graphical Interchange Format, or GIF, resembles PNG-8 in many ways.

   It's universally accepted by every browser. It's established, having been around since 1987 (somewhat younger, PNG debuted in 1996). GIF uses color tables. It's compression is slightly less efficient, on average, than PNG. And it can make its pixels completely transparent.

   Where GIFs shine is their ability to hold more than one "image" per file, and show them sequentially. That is, GIF supports animation. Animated GIF files usually find their way into content more than design, as their animations can't be started and stopped – they're not truly interactive. That and their slightly-higher file sizes mean that designers often prefer PNG over GIF for flat-color images.

   If you need simple animations, GIF is the way to go. Otherwise you're probably better off with PNG-8's slightly more efficient compression scheme for simple images (like flat-color illustrations) or JPEG for complicated images (like photos).

Squeezing Every Byte

Compressing image files – the act of reducing file size by eliminating redundant data or altering the image for easier downloads – is crucial to making websites load quickly. Fast websites, in turn, earn more users.

Most image editors like Photoshop, Sketch and Pixelmator export compressed, web-friendly files without fuss. But they're not always ideal. Other tools can help compress images even further.

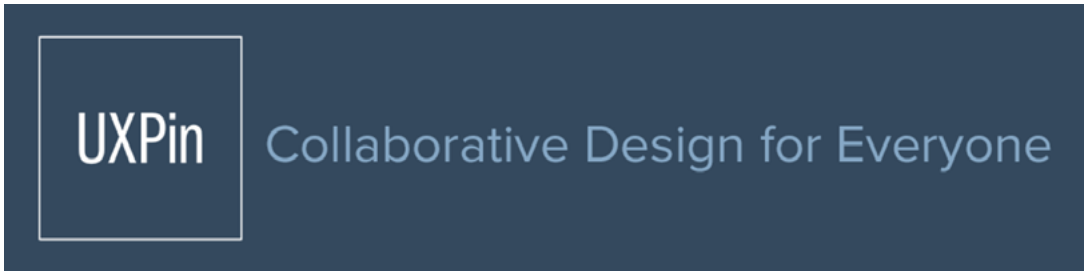
Compression Services

1. Compress JPG
   As the name implies, this free, online service by Mediafox Marketing takes extra bytes out of any JPG file without sacrificing quality.

Compress JPG slimmed down the image above, saved in Photoshop at 70%, from 217KB to a svelte 160KB. Quality remains untouched.

2. TinyPNG

   Another free service – this one from voormedia – squeezes 8-bit and 24-bit PNG files for faster load times.



   TinyPNG reduced the above image from 16KB to 12KB by eliminating nearly redundant colors in its color table – without affecting its quality.

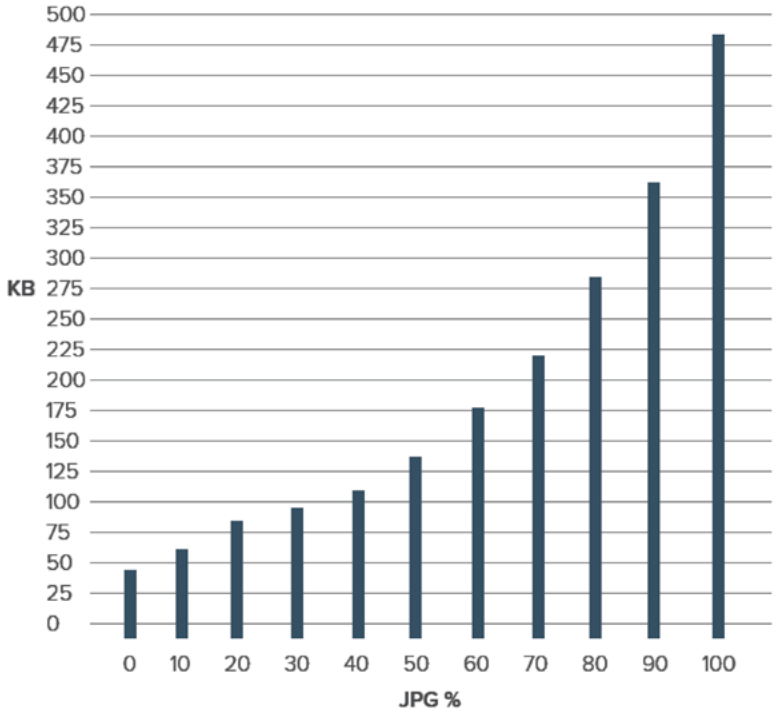
3. How Much is Too Much... or Too Little?

   Although it varies per image, our goal is to get the best quality image in as few bytes as possible. At some point, we trim too much out.

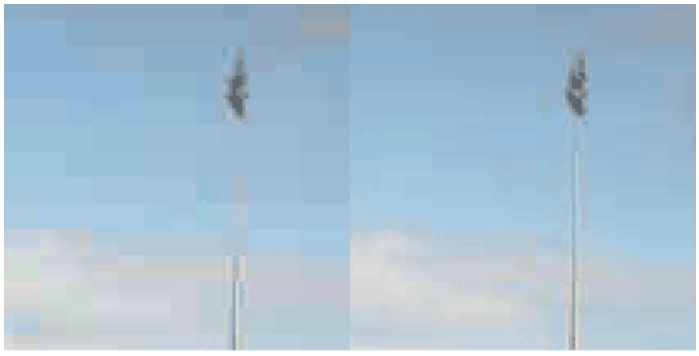
   But how much? Is there a sweet spot for compression?

   For JPGs

   To find out, we saved the crowd photo above with increments of JPG compression. Results ranged from 45KB at 0% compression to 479KB at 100%. Contrary to what the term suggests, remember that the highest compressed JPGs have the highest quality (and largest file size).



This chart shows a dramatic decrease in the higher compression ranges. Just dropping the quality from 100% to 70% reduced the file size by almost half – a real bargain. The lower compression percentages, though, didn't see as much of a difference. Although bytes fell from 0 – 20%, we saw quality drop too quickly to be worth the savings.



0% JPG 30% JPG

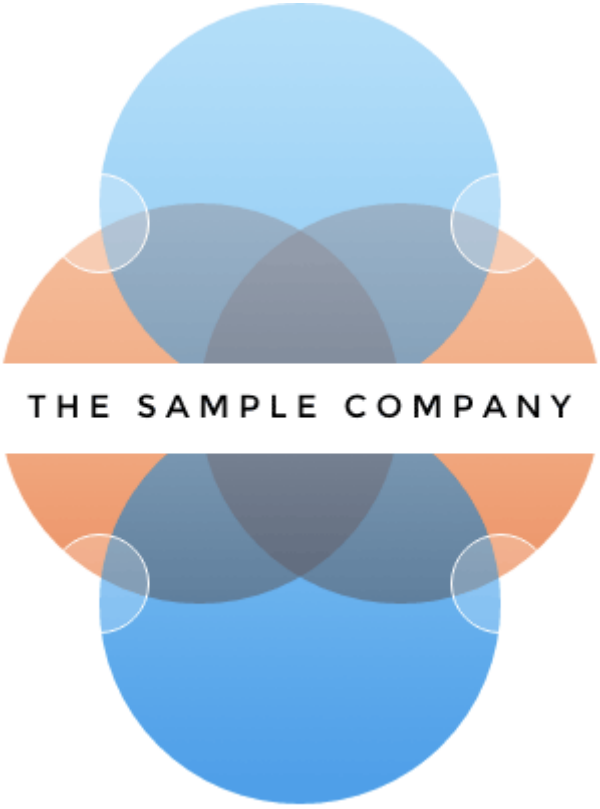
The image was 45KB at 0% compression (left) and 94KB at 30% compression (right). Although it halved the file size, the increase

in artifacts, or blocky areas where JPG compression takes effect, wasn't worth the savings.

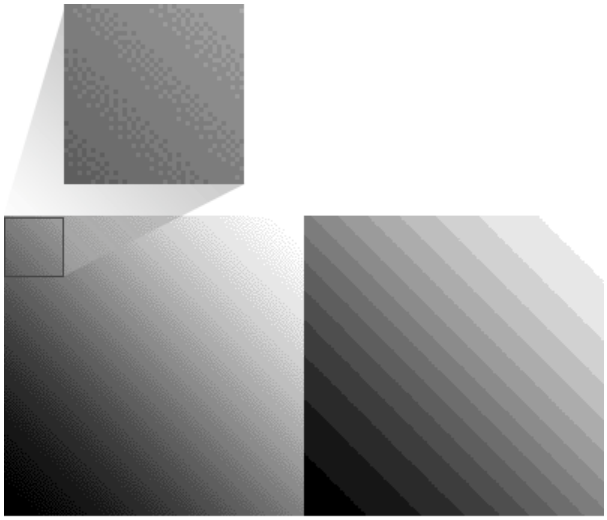
Best practice: Do not compress JPG files higher than 70%, or lower than 20%. This is a guideline rather than a hard rule, but we've found it the 20–70 range covers most cases.

For PNGs

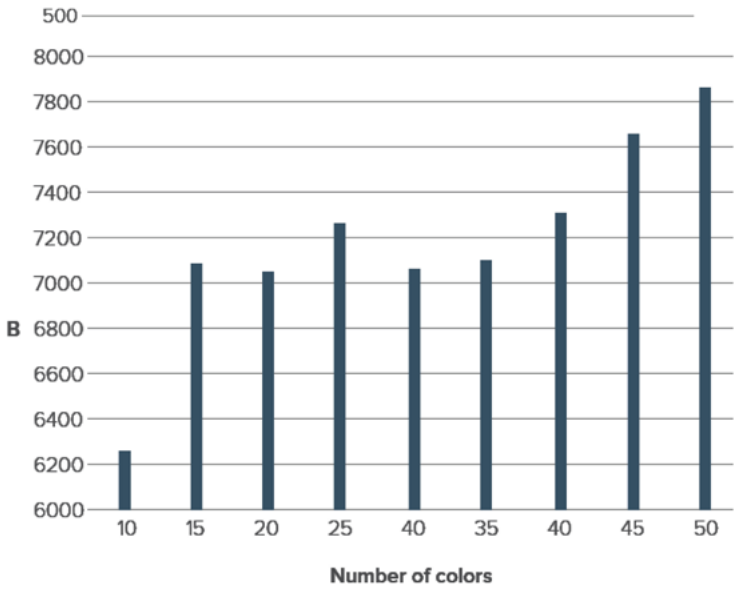
The story gets more complicated when we look at PNG files. We ran the same experiment on this graphic:



Notice that these colors aren't strictly flat. There's a fine gradient over the entire composition. To account for that, we need dithering: a pattern of dots that simulate subtle gradients.



Unlike JPG, the PNG format doesn't use percentages. The number of colors in its color table determine its quality and, to an extent, its file size. With 88% dithering, the results were, well...



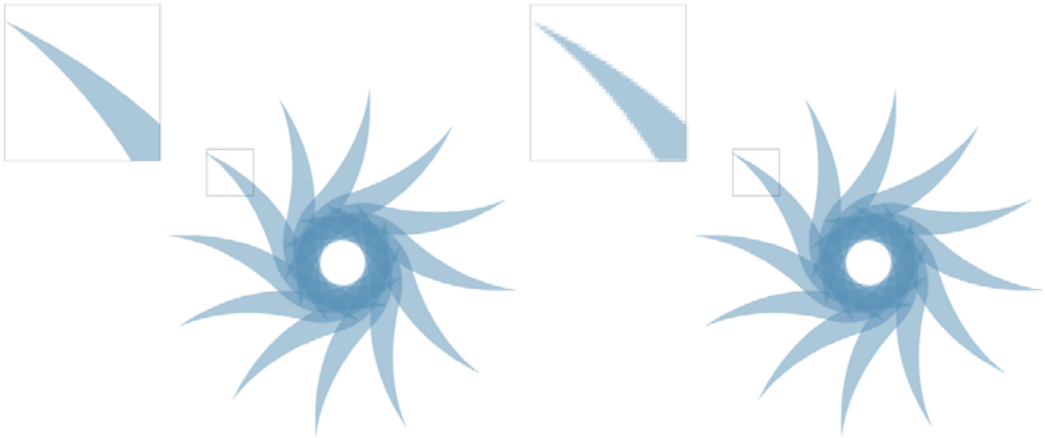
In general we saw a relationship between the number of colors and file size. But not much. Photoshop struggled to find the best patterns with limited color tables. In fact, 40 colors had about the same number of bytes as 25, meaning the same file size but with much higher quality.

Best practice: For best results when squeezing every byte out of a PNG, the best approach is to experiment with different color tables. Unfortunately the right amount is a subjective matter that varies per image. When it looks “right” is up to you.

4. SVG

   Scalable Vector Graphics, or SVG, use lines instead of pixels – vectors instead of raster images – to display line art. SVGs are actually a form of XML, easily created in programs like Inkscape and Adobe Illustrator.

   SVG files can generate gradients without dithering, and scale up to fit containers of any size from older smartphones to widescreen TVs. They don’t lose resolution because, as vectors, browsers connect the dots on the fly. They can be animated with JavaScript. And instead of files that must be downloaded separately, which ties up time and server power, SVG can be embedded right in HTML documents.



Above: vector art (left) scales up well. On the other hand, raster art (right) looks blocky and pixelated.

But like PNGs and GIFs, SVGs suffer as images become more complex. They're terrible for photos and grow quickly in file size as they gain points and curves.

If you're looking for the flat 2.0 look, with its sharp lines and gentle gradients, and aim for recent browsers (IE8 is out of luck), SVGs are the way to go.

Code Considerations

Aside from image files themselves, we can do lots with code to make pixels respond well to different situations.

1. Essential CSS Properties
   One of the most common – and most reliable – solutions is to set a bit of CSS:

   img { max-width: 100%; }

   This selector and property makes most images fit into their containers. For example, if a media query sets a wrapper to 300 pixels in width, then no image inside that wrapper will exceed 300 pixels. This technique has excellent support across modern browsers, which is why you'll find it in many responsive websites today.

2. Future HTML Image Elements
   Today we're limited to background images with CSS and the <*img*> element. But when (and if) implemented, a new technology will

go a long way to making images as responsive as page layouts in responsive web design.

The experimental <*picture*> element contains one or more <*source*> child element that uses media queries to declare when they should load. Browsers replace the src attribute of an <*img*> element within <*picture*> with the relevant source, if any. For example:

```
<picture alt="Descriptive text fallback">
  <img src="sample-default.png">
  <source srcset="sample-large.png" media="(min-width: 640px)">
  <source srcset="sample-small.png" media="(max-width: 639px)">
</picture>
```

The code above would replace *sample-default.png* with either a large or small variation, depending on the image's container's width.

As a bonus, browsers that don't support <*picture*> will still read the default <*img*> element as normal. That's good news because, at the time of this writing, these elements are not universally accepted – in fact, few modern browsers support them today. But support for <*picture*> and <*source*> is growing, and smart designers will keep an eye out for their usage in the future.

3. Browser rendering

   Sometimes the best image is none at all. Modern browsers are capable of rendering their own graphics, including gradients, animations, bezier vectors, shadows and geometric shapes. With a little creativity, we can even make stripes.

Creating images in browsers puts the burden of visuals on the user's end. It doesn't require downloading any image files, which saves bandwidth and time. But it does require the browser to have certain capabilities.

Best practice: Use browser rendering for aesthetics, like background colors and fancy borders, but don't rely on it for content. Always test your designs for readability without fancy CSS3 tricks, or even without CSS at all, to see how gracefully it degrades on less-capable browsers.

Implications for UI Design

From icons to backgrounds to content, there's no doubt that images are crucial for good user interfaces. But responsive images face many challenges including bandwidth concerns and sizing issues.

In a perfect world we'd have the ability to either crop an image for smaller viewports, focusing on the most important parts, or the ability to upload different images for different breakpoints. It *is* possible to do so. Workarounds exist, and we look forward to <picture> and srcset. Until then the best-practice approach is to test your images at various sizes to make sure they're readable on various devices and browsers.

Designers concerned with responsive web design must consider images in their work. Using the right file format, optimizing compression

(but not too much) and watching future technologies go a long way to making websites load quickly and look great on screens of any size, resolution or orientation.

It all comes down to one question: what serves the user best?

The Technical Side of Responsive Design

It's time to delve into the underpinnings of responsive web design.

As a technical medium, RWD requires a working knowledge of the code that makes it work. Designers who ignore the basics fail to understand what browsers can and can't do.

Cascading Stylesheets, or CSS, is the language that makes responsive web design possible. It's a series of directives that spell out how browsers should display data depending on certain conditions like browser width. Now, understand that while most modern browsers obey the same rules in the same way, CSS rules are suggestions.

Let's review the fundamentals. CSS works with *selectors* and *properties.*

- **Selectors**: HTML affected by the CSS. Examples: HTML <*p*> elements, <*li*> elements and <*div*> elements.
- **Properties:** what the CSS will change on a given selector. Examples: Text color, border and padding.

- **Values:** the change itself. Examples: Red, 10px and sans-serif are three values.

Now let's put this together. *"div { color: #333; }"* has one selector, one property and one value.

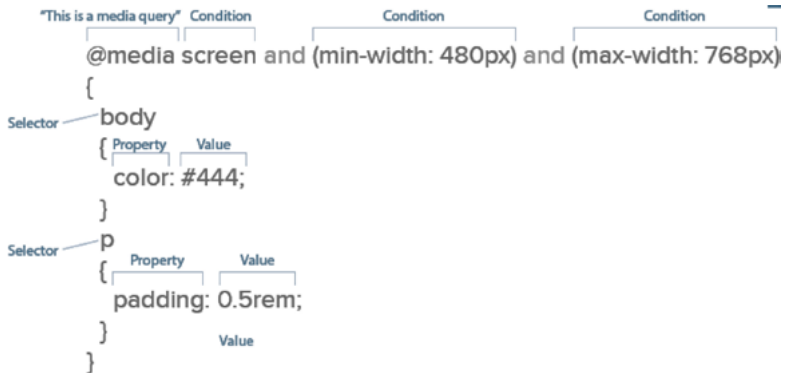
Media Queries Are the Technical Heart of Responsive Web Design

To account for different browsing conditions, responsive web design dictates that different rules must take place under different circumstances, usually the browser's or device's screen (the "viewport") width.

How do we define these rules? Media queries. Media queries are CSS commands that determine under what conditions other CSS selectors take effect.

For example, one media query might tell a browser, "pay attention to rules 1–10 when the screen is up to 320 pixels wide," while another might say, "pay attention to rules 11–20 when the screen is 321 pixels wide or greater."

Media queries are easy to identify: they begin with “@media”. Browsers read the CSS rules (e.g. selectors) listed between the media query’s { curly brackets }.



Above, CSS says to make all text red on visual devices, not screen readers.

Media queries can build upon each other. For example:

@media screen and (width: 320px) { *body* { *color: red;* } }

Rules take effect on visual devices that display exactly 320 pixels horizontally.

The different minimum and maximum widths that media queries use are called *breakpoints*. A query that specifies (max-width: 768px) would change layouts when the viewport measures 0–768 pixels wide. Oddly, nothing actually “breaks” at that point. he term simply means that new rules will take effect within a given range, in this case 0–768 pixels, or the width of an average tablet.

There’s no technical limit to the number of conditions a media query can display. This query is perfectly valid:

@media screen and (min-width: 480px) and (orientation: landscape) and (aspect-ratio: 4/3) and (color: true) {... }

It says that any color screen at least 480 pixels wide, held in landscape position, with an aspect ratio of 4 to 3 that's exactly 320 pixels wide should take on certain properties.

The two most popular conditions are minimum and maximum width, the upper and lower limits a browser window can be to take on the given properties. But we can manipulate other important properties.

Resolution, for instance, suggests at what quality we should create our graphics. Screens capable of high-density graphics, e.g. Apple's retina screens or high-res Android screens, will show text and vector art with crisp lines that make regular-density images look fuzzy.

"High-density images" are those that use more device pixels (actual dots a browser can display) vs. those defined in CSS. That means for an image to look its best, it needs four times the usual number of pixels. Something displayed at 300×300 pixels on a small screen actually needs 600×600 pixels total – when our media queries detect high-res screens.

If you're curious, here are additional properties we can control with media queries:

- **Aspect-ratio:** a comparison of a browser window's width and height.

- **Color:** whether a device has a color screen or not.
- **Color-index:** the number of colors a screen can display.
- **Device-aspect-ratio:** a comparison of the screen's width and height.
- **Device-height:** the number of pixels a displayed vertically by a device.
- **Device-width:** the number of pixels a displayed horizontally by a device.
- **Height:** the number of pixels displayed vertically by a browser viewport screen.
- **Monochrome:** how many bits that pixels in a grayscale screen uses.
- **Orientation:** whether a user is currently holding the device horizontally or vertically.
- **Resolution:** the number of pixels per inch or centimeter displayed by a screen.
- **Width:** the number of pixels a browser viewport uses horizontally.

As web layouts become more sophisticated, orientation and aspect ratio will help us decide how much to show users at a glance. The "fold" may not be relevant as users scroll, but what they see when they stop scrolling needs to be as self-contained as possible.

For now, though, width and resolution are most useful for responsive web design. Since contemporary site design lets us scroll up and down, width determines the available space a layout can use.

1. Best Practice According to the Experts

   It might not surprise you to learn that Google, whose search engine reads a page's information and ignores viewports, recommend setting media queries by content, not specific devices. That makes sense as new devices appear every month, meaning device-based CSS would need constant updates.

   Here's where media queries help us most: they allow us to plan for browsers based on content and capability, not expectations. That is, we can tell browsers that our designs look best under ranges of conditions, like 0–300 pixels vs. 301 – 600 pixels, and write our code to suit our needs rather than what the browser requires.

   Google also recommends adopting a mobile-first approach which, in addition to focusing on essential content that users want, encourages designers to use the fewest breakpoints possible. That means easier troubleshooting during development and maintenance later on.

   If you'd like to learn more techniques for CSS media queries, we recommend the following resources:

   - Responsive Layouts Using CSS Media Queries
   - Create a Responsive Web Design With Media Queries
   - Introduction to Responsive Web Design: Pseudo-Elements, Media Queries, and More
   - Quantity Queries for CSS

Compressing Resources for Faster Load Times

Like images, we can shrink CSS files to use as few bytes as possible. Smaller files require less time to load, making websites faster and keeping users from wandering away.

Many techniques exist to keep files small, but we recommend starting with a few basics.

1. Minification

   With good spacing, CSS and HTML files are easy for designers to develop and maintain. But browsers don't care about spaces between elements, properties and selectors. Extra spaces mean extra bytes, so we want to minimize those spaces (hence the term).

   Before:

```
body {
  background: #fff;
  color: #444;
}
p {
  padding-bottom: 1rem;
  margin: 0;
}
```

   After:

```
body{background:#fff;color:#444}p{padding-bottom:1rem;margin:0}
```

Tools like CSS Minifier, CSS Compressor, JS Compress and HTML Minifier by Will Peavy let you quickly strip away excess spaces from production copies of your code, while keeping the original lets you easily edit the original.

Best practice: check your code and always minify copies of your HTML, CSS and JavaScript files before uploading them to the live website.

2. GZIP

   Like LZW compression used in PNG and GIF files, GZIP scans files for redundant bytes. This is a deeper compression technique than minification. Google reports that this works best with text-based files like HTML, CSS and JavaScript.

   Best practice: zip files after minifying them to make sites load faster, especially on smaller, less-capable devices.

HTTP requests

An often-overlooked means to speed up your sites is to reduce the number of files that a browser must download per page view. Simply put, the fewer files to download, the faster browsers can display a page.

Each request can take from 200 ms (milliseconds) to several seconds. That adds up quickly since (as we explained in *Interaction Design*

Best Practices) users start losing their feeling of control after a 1-2 second delay.

Once again, the advantage here is speed, especially for small devices with limited bandwidth or slow connections.

Best practice: whenever possible, save time by combining your minified CSS and JavaScript files. Every few hundred milliseconds counts.

Takeaway

Technical details like these are just as important to responsive web design as the visuals. Designers who neglect performance and understanding media queries risk delivering poor experiences to their users.

Putting It All Together: The Mobile-First Workflow

All of this best-practice advice is nothing without some common sense. But achieving that mindset requires us to hone our thinking into a flexible mindset. We need to reassess what we think we know, to design for small devices first, to test on real devices, and to realize that our designs are not fixed, but fluid.

Question Your Assumptions

Thinking responsive means adapting our workflow – but it's mostly common sense. The trick is to question your assumptions about your users. For example, frequent assumptions include:

- Everyone uses Chrome.
- Sizing a browser down is the same as using a smartphone.
- Users share your business goals.
- Everyone supports JavaScript.

- On-screen keyboards won't interfere with the ability to use a form.
- Everyone's computer is as fast and capable as yours.
- Everyone has fast, reliable wifi or LTE.
- Everyone will view your work in the same lighting conditions as your workplace.
- SEO is someone else's problem.

Collaborate From the Beginning

Responsive design is especially hard to just hand off because it is technically involving.

Based on Brad Frost's atomic design framework, map out your overall layout in a wireframe or lo-fi prototype, then start refining details of components reused through the design. Involve developers so you can start laying out the groundwork as early as possible.

Test, test and test again as you design. Use real devices – don't assume your mouse cursor is the same experience as a fingertip. Invite developers to any usability testing sessions so they can start to grasp the technical implications of the necessary revisions.

If you're a designer who handles the development, you can try the following tactics as well:

- Occasionally develop on a smaller screen. Forego your 24" screen for a laptop.
- Identify problems by playing with different amounts of content. How does the layout react to too much or too little content?
- Try it without CSS and JS. What breaks? What's frustrating? Or does it actually make things easier?

Helpful resources:

- See Your Site Responsive
- Google Mobile Friendly Test
- Responsive Web Design Testing Tool
- Adaptive Images
- W3C validator
- Adobe Edge Inspect
- Responsive Workflow

Embrace Mobile-First Design

The mobile-first approach is exactly as it sounds: designing for the smallest screen and working your way up. It is one of the best strategies to create either a responsive or adaptive design.

- **The mobile-first approach is a tenet of progressive enhancement.** It is the ideology that mobile design, as the hardest, should be done first. Once the mobile design questions are answered,

designing for other devices will be easier. What it boils down to is that, the smallest of the designs will have only the essential features, so right away you have designed the heart of your UX.

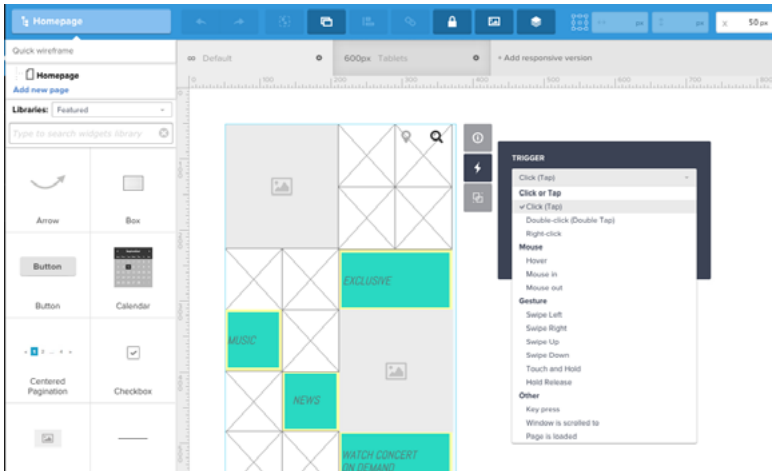


Photo credit: UXPin

- **The opposite approach is graceful degradation.** This incorporates all of the complexities right from the start, then strips them away later for smaller devices. The problem with graceful degradation is that when you build the all-inclusive design right from the start, the core and supplementary elements merge and become harder to distinguish and separate. The entire philosophy runs the risk of treating mobile design as more of an afterthought since you're "cutting down" the experience.

We, along with many others, strongly recommend progressive enhancement with a mobile-first approach.

Mobile-First = Content-First

If your site is good on a mobile device, it translates better to all devices. More important, though, is that the mobile-first approach is also a content-first approach. Mobile has the most limitations, screen size and bandwidth to name a few, and so designing within these parameters force you to prioritize content ruthlessly.

The mobile-first approach organically leads to a design that's more content-focused, and therefore user-focused. The heart of the site is content – that's what the users are there for.

One caveat, though, is that mobile users sometimes require different content than desktop users. Device-specific content can be gauged by considering context – what, in a given situation and a given environment, will your user appreciate more. The best way to plan ahead for these is creating user scenarios.

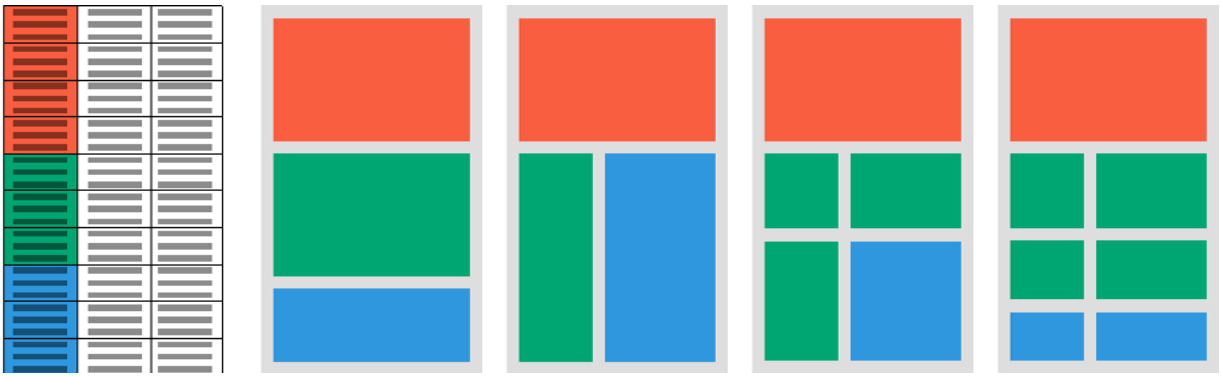


Photo credit: UXPin

Another advantage to mobile-first approach is that the small-screen breakpoints can better fit around the content. Again, the alternative is worse: having to squeeze an already plump design into a tiny frame-

work. But with the mobile-first approach, the breakpoints develop naturally around content, so you don't need any awkward edits.

The Mobile-First Process

We'll describe a process that helps our designers at UXPin.

As usual, wireframing is a recommended early step to most efficiently structure your layout. When wireframing or prototyping, we use the responsive breakpoint menu streamlines the process of moving to different screen sizes, starting with the smallest.

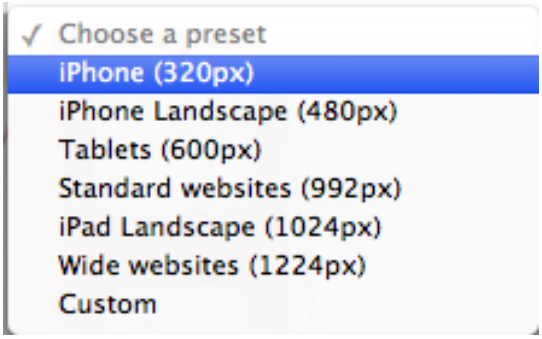


Photo credit: UXPin

These presets layout the proper screen size for you, so you can wireframe keeping only the content in mind.

Our procedure follows these steps:

1. **Content Inventory** – This is a spreadsheet or equivalent document containing all the elements you want to include.

	Navigation title	Page title	Files	Last updated	Owner	Comments	Delete?
0.0	Home	Wine Tasmania					
1.0	Wine Tasmania					No page at this level - displays 'History'	
1.1	History	History					
1.2	Touring Tasmania	Touring Tasmania					
1.3	Touring Links	Touring Links					
1.4	Wine Industry Tasmania	Wine Industry Tasmania					
1.5	Industry Statistics & Info	Industry Statistics & Info					
1.6	Investment	Investment					
1.7	Partners	Wine Industry Tasmania Partners					
2.0	The Wine Route					No page at this level - displays 'Overview'	
2.1	Wine route overview	The wine route					
2.2.0	Tamar Valley Wine Route	Tamar Valley Wine Route					
2.3.0	Southern Wine Region	Southern Wine Region					
2.4.0	East Coast Wine Region	East Coast Wine Region					
2.5.0	North West Wine Region	North West Wine Region					
3.0	Latest News	Latest News				No content on page	
4.0	Events					No page at this level - displays 'Overview'	
4.1	Overview	Events				No left-nav	
4.2	Booking	Event booking				No left-nav	
4.3	Privacy Policy	Privacy Policy				No left-nav	
4.4	Security and Refunds	Security and Refunds				No left-nav	
5.0	Members	Wine Industry Tasmania Members					

Photo credit: Maadmob

2. **Visual Hierarchy** – Prioritize the elements in the content inventory and determine how to display the most important elements prominently.

3. **Design with the smallest breakpoints and then scale up** – Now that you know your most important content, you can get started with the smallest viewport. Build the mobile wireframe first, then use that as the model for larger breakpoints. Expand the screen until there's too much white space, then subtract it down until it feels "just right". If you're using UXPin, you can also add interactions simultaneously, turning your responsive wireframe into a lo-fi responsive prototype.

4. **Enlarge touch targets** – Fingers are much wider than pixel-precise mouse cursors, and so need larger elements on which to tap. At the time of this writing, Apple recommends 44 points square for touch targets. Give hyperlinks plenty of space, and slightly enlarge buttons, and make sure that

5. **Don't count on hovers** – It almost goes without saying, but designers often rely on hover and mouseover effects in their inter-

active work. If you're thinking mobile-friendly, don't. There is no hover control for fingertips yet.

6. **Think "app"** – Mobile users are accustomed to motion and a modicum of control in their experience. Think about off-canvas navigation, expandible widgets, AJAX calls, or other elements on the screen with which users can interact without refreshing the page.

7. **Avoid large graphics** – Landscape photos and complex graphics don't display well when your screen is only a few inches across. Cater to mobile users with images that are readable on hand-held screens.

8. **Test it in a real device** – Nothing beats discovering for yourself how usable a website is (or isn't). As you design, periodically step away from your desktop/laptop computer and load up your product on a real phone or tablet. Tap through pages. Is the site easy to navigate? Does it load in a timely fashion? Are the text and graphics easy to read?

Test your design with at least 5 real users. When we're designing responsively in UXPin, we'll use the app's SMS or QR code function to send the prototype to our mobile device. After completing any necessary tweaks, we'll then create user tasks and use the remote usability testing feature in UXPin to validate the design. Since the entire session is recorded, we can see all the clicks and feedback for future reference. You can either run your own similar test or use a paid service like User Testing.

1. A Mobile-First Design Tutorial

   Given that different devices need different layouts based on their screen size and orientation, it makes sense to design multiple arrangements for your users. Luckily you can make your own responsive or adaptive variations right in UXPin.

   We'll create an example and describe how to scale up content from a smartphone to the tablet and desktop views.

   Set your content priorities

   A "mobile-first approach" differs from "desktop-first" in that we add information to each progressively-larger layout rather than cut away as we design smaller. Thinking mobile doesn't mean eliminating information. It means sorting information into primary, secondary and tertiary content.

   In this example, we know that the home page should have certain elements, like the company's name and links to products. A blog post wouldn't hurt either. But like we said, not everything will fit into a smartphone view, so we set priorities based on what will achieve the site's goal: selling bikes:

   1. The newest model bike
   2. The best-selling bike
   3. "Find your perfect ride" CTA
   4. Company name and hero image
   5. Navigation
   6. Search
   7. The second-best-selling bike

8. Gift certificates
9. A testimonial
10. The latest blog post

Based on that ordered list, we can create with the confidence that our work will solve a design problem of getting sales.

On setting breakpoints

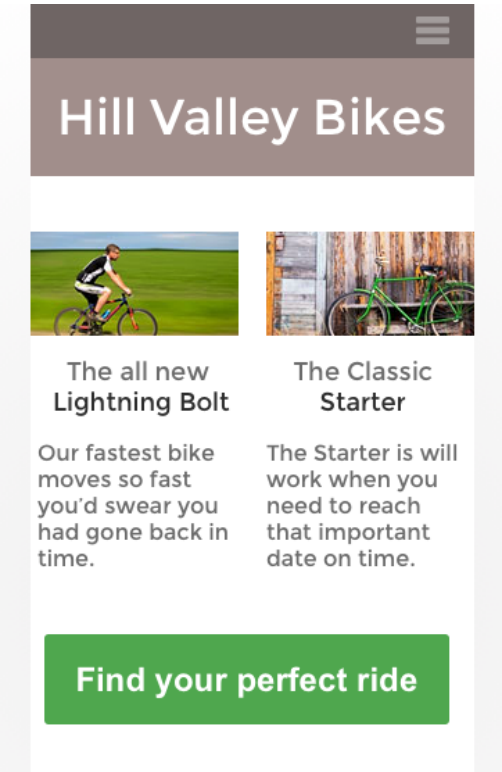
At what point should we set breakpoints? As we've seen, the answers vary quite a bit. But there are some rules of thumb.

- **Use text as your guide.** As Google recommends, the smallest breakpoint should be about 10 (English) words wide. Plan for a minimum of 30 rems on smaller screens…
- **Start small.**… then settle on breakpoints for increasingly larger screens. This not only helps you focus content, but also encourages designers to reach users on any platform first, then add (optional) features for more capable browsers and devices. It also helps serve both users with special needs and concentrate on SEO.
- **Be adaptable.** Don't set fixed-width containers for content. Instead, use max-width to let text and images adapt to *ranges* of viewports. You never know when one screen will use, say, 420 pixels and other 480.

Choosing breakpoints isn't about hitting device screen widths. Best practice demands that we design for what we know, and that

means the text, images and media for which users visit our sites to begin with.

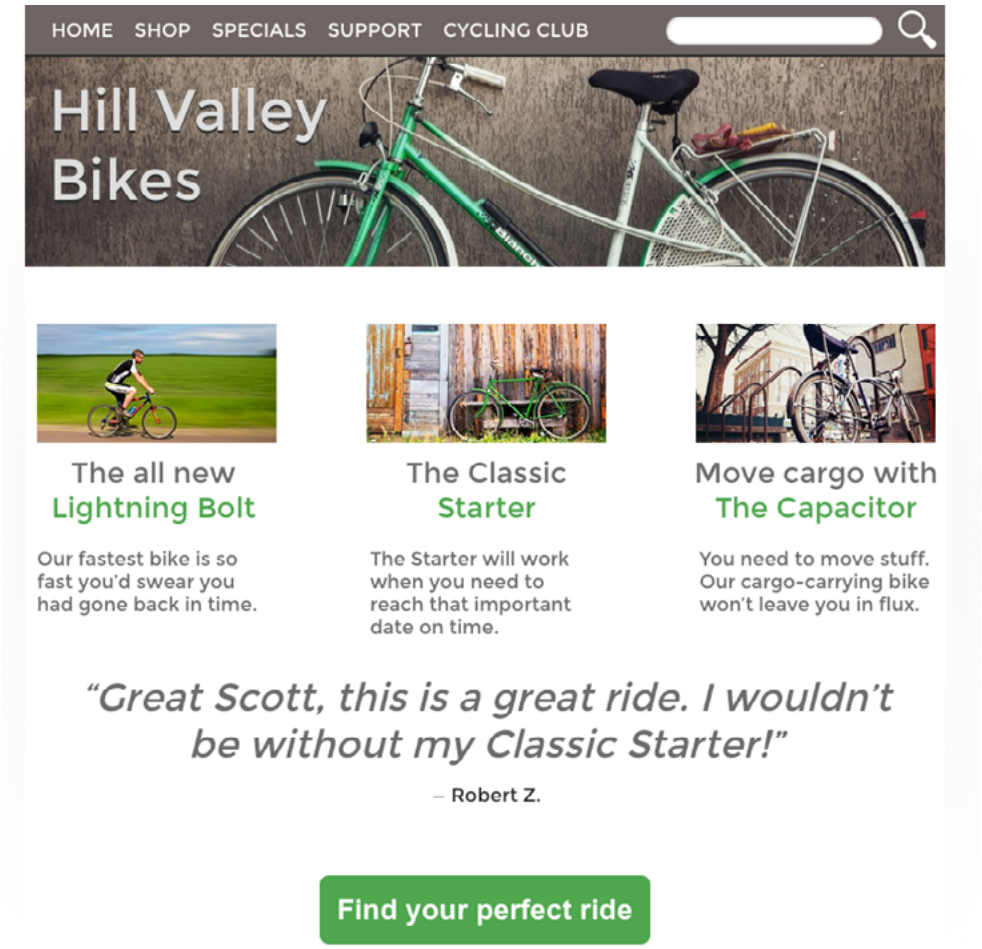
Smartphone View:



How much do users need?

Thinking mobile-first forces us to think about what's *really* important. In this smartphone view, the top-selling bike and newest model will lead directly to sales, so can we leave other items – such as gift certificates, a less-popular model, the latest news – for inside pages. The final call to action is especially prominent and easy to hit with a single tap of the finger.

Tablet View:

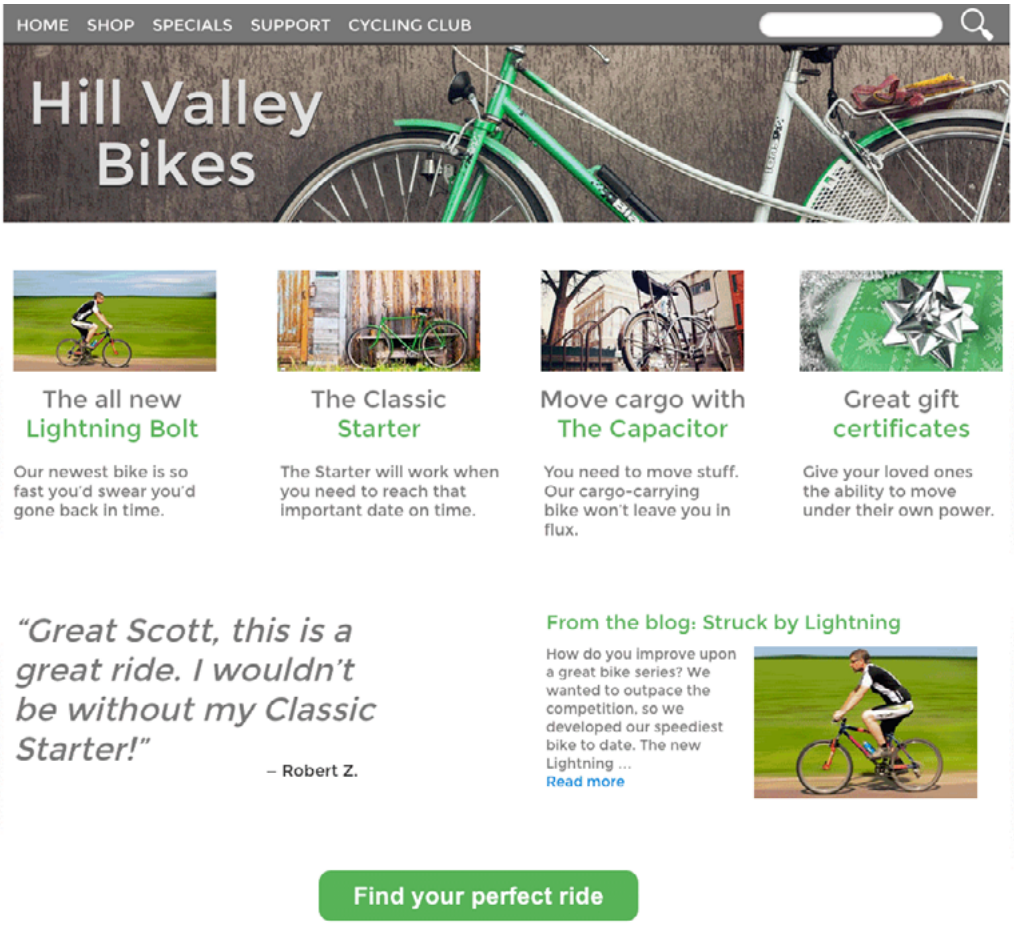


As we design for a tablet-sized view, we're better able to add secondary information like additional products (e.g. "The Capacitor"). We can also expand the navigation at the top of the page and add content that encourages sales without actually leading to them – namely, the testimonial.

Because more options are available, this can be surprisingly more difficult than deciding what to include in a smartphone UI. The difference between secondary and tertiary elements is a blurry line, and temptation is strong to include everything.

Resist the urge. Use the ordered content list. Like smartphones, space is still limited.

Desktop View:



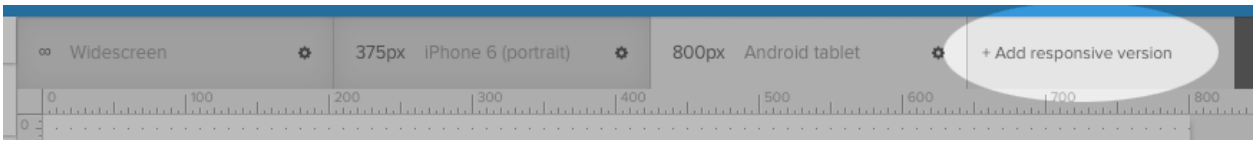
Finally, the desktop view can support as much information as you decide is important. This is where the home page can accommodate all of the information you see fit, whether or not it fits. Notice some of the additional content we've included:

- Gift certificates
- Customer testimonials
- Blog post exploring the newest Lightning Bolt bike

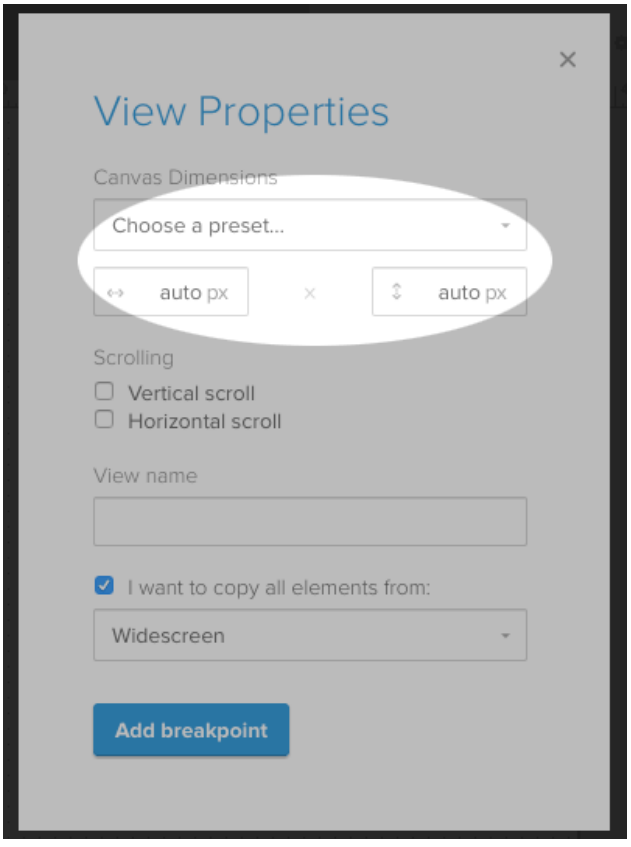
Design device-appropriate layouts yourself

If you're using UXPin, it's fairly easy to create different layouts for these views.

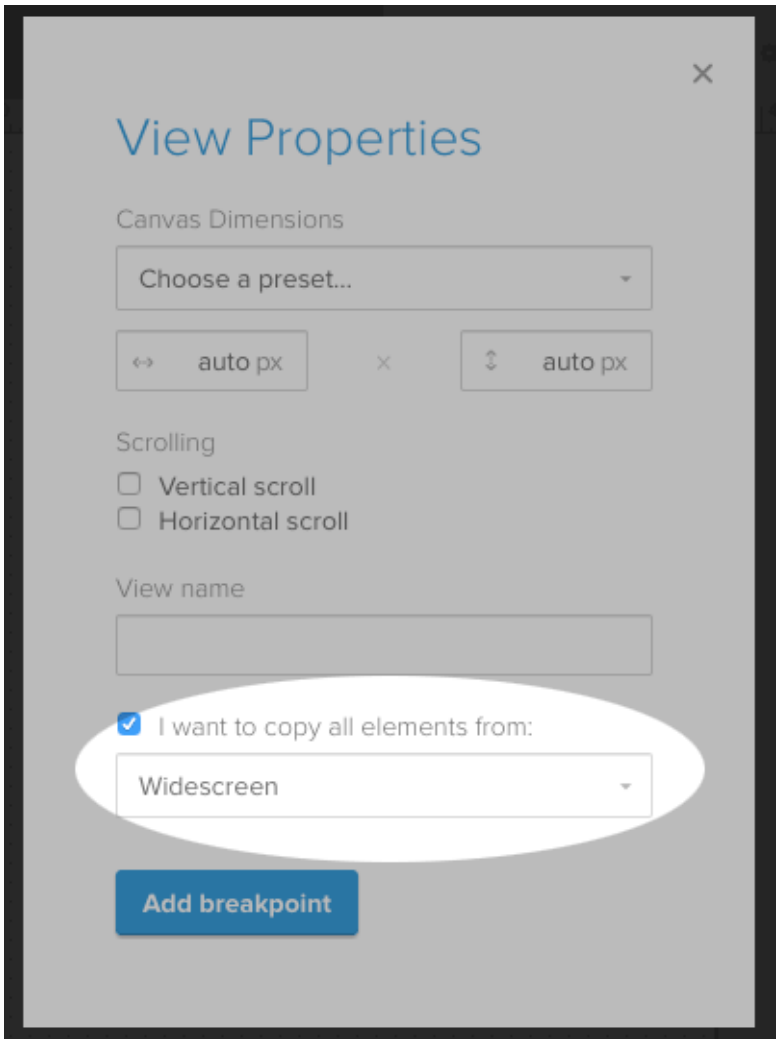
1. Open a UXPin prototype.
2. Tap "add responsive version" at the top of your prototype's canvas.



3. Choose a preset size or enter your own dimensions.



4. You don't have to recreate everything from scratch. Choose a size from which to copy your design's elements.

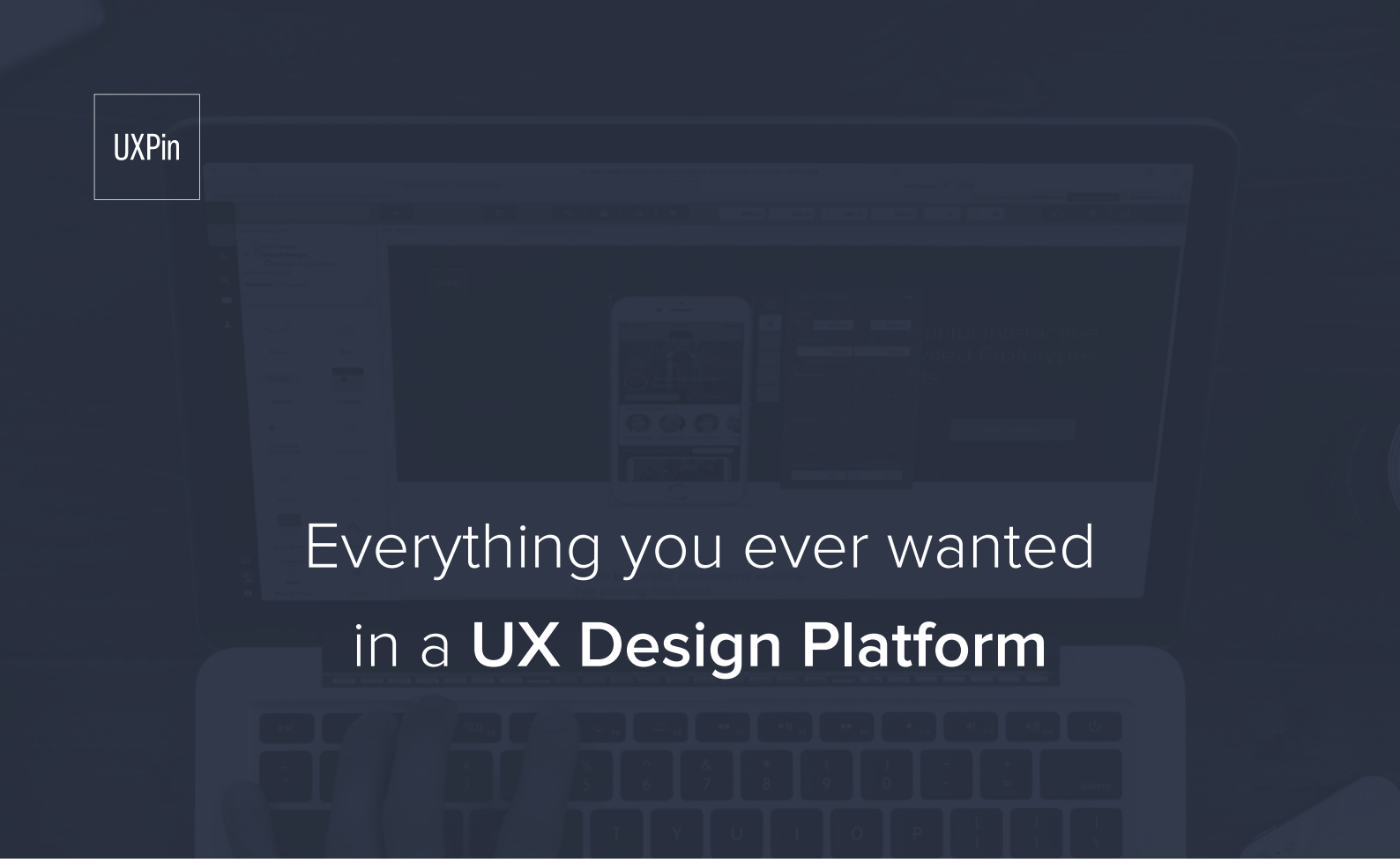


And that's it. Switch between breakpoints by tapping the different sizes above your canvas, and adjust each to suit your needs.

Going forward

Nothing beats practice. Test it. Seek outside opinions, and not just from other designers – get folks to try your work. Collaborate with your team. And stay flexible in your thinking to make your designs work well on browsers of any configuration.

UXPin



Everything you ever wanted in a **UX Design Platform**

- Complete prototyping framework for web and mobile
- Collaboration and feedback for any team size
- Lo-fi to hi-fi design in a single tool
- Integration with Photoshop and Sketch

Start using it now!

www.uxpin.com